

مركز البحوث

إدارة وتطوير مشاريع البرمجيات

تأليف:
د. عبدالوهاب كامل السماك

بسم الله الرحمن الرحيم



مركز البحوث

إدارة وتطوير مشاريع البرمجيات

تأليف

د. عبدالوهاب كامل السماك

١٤٢٩هـ - ٢٠٠٨م

بطاقة الفهرسة

ح معهد الإدارة العامة، ١٤٢٩هـ.

فهرسة مكتبة الملك فهد الوطنية أثناء النشر.

السماك، عبدالوهاب كامل

إدارة وتطوير مشاريع البرمجيات/ عبد الوهاب كامل السماك

الرياض، ١٤٢٩هـ

٣٢٨ صفحة ١٧ × ٢٤ سم.

ردمك: ٥-١٦٩-١٤-٩٩٦٠-٩٧٨

١- البرمجة (حواسيب) ٢- تخزين واسترجاع البرمجيات

أ. العنوان

ديوى ٠٠٥,٣٦٩ ١٤٢٩/١٠٩٣

رقم الإيداع: ١٤٢٩/١٠٩٣

ردمك: ٥-١٦٩-١٤-٩٩٦٠-٩٧٨

إهداء

إلى والدتى وروح أبى

إلى زوجتى وأولادى

إلى أساتذتى وزملائى

وإلى طلاب العلم فى الوطن العربى

قائمة المحتويات

الموضـــــوع	رقم الصفحة

الموضـــــوع	رقم الصفحة

الموضـــــوع	رقم الصفحة

الموضـــوع	رقم الصفحة

الموضـــــوع	رقم الصفحة

الموضــــوع	رقم الصفحة

الموضـــوع	رقم الصفحة

قائمة الأشكال

تابع - قائمة الأشكال

تابع - قائمة الأشكال

تابع - قائمة الأشكال

تابع - قائمة الأشكال

تابع - قائمة الأشكال

تابع - قائمة الأشكال

مقدمة:

بسم الله والصلاة والسلام على رسول الله وعلى آله وصحبه وسلم، وبعد.

لقد شـهدت العشـرون عاماً الماضية طفرة كبيرة فى معظم البلـدان العربية فى اسـتخدام تقنيـة المعلومات فى مجالات الحيـاة المختلفة مثل المجـالات الاقتصادية والصناعيـة والزراعيـة والتعليمية والصحية والاجتماعيـة والخدمية والأمنية وحتى المجـالات الإعلامية والترفيهية التى تحتوى على عدد كبير جداً من تطبيقات الأعمال التى تحتوى بدورهـا على مجموعة كبيرة من إجراءات العمل ويتم تطبيقها على كمية كبيرة من البيانات بواسـطة مجموعة من المستخدمين من خلال نظم تشغيل مختلفة. يتطلـب ذلك توافر العديد من الأسـاليب والأدوات والموارد التى تدعم عمليات تطوير البرمجيات المناسـبة لتلك التطبيقات التى يجب أن تتميز بإمكانية الاتسـاع وسهولة الصيانة بحيث يمكننا إضافـة متطلبات جديدة فى أى وقت أو التعديل فى المتطلبات الحالية.

ولأن عمليات تطوير البرمجيات للأنظمة كبيرة الحجم تستغرق وقتاً كبيراً ويشارك فيها عدد كبير من الأفراد وتسـتخدم العديد من الموارد المادية، فإنها تحتاج إلى إدارة جيدة للسيطرة على تلك العوامل المؤثرة فى نجاح مشروع البرمجيات، حيث تهتم إدارة مشاريع البرمجيات بعمليات التخطيط والمراقبة والتحكم فى التغيرات التى تطرأ على المشروع وكذلك السيطرة على الموارد المستخدمة. وتؤدى غالباً الإدارة الجيدة للمشروع إلى إنتاج البرمجيات المطلوبة فى الوقت المحدد وبالجودة المطلوبة وفى حدود الميزانية المقدرة للمشروع، مما يتطلب استخدام مجموعة من الأساليب والأدوات الحديثة التى تدعم عمليات إدارة وتطوير وإنتاج مشاريع البرمجيات.

ولقد ظهرت فى السـنوات القليلـة الماضية العديد من الأسـاليب والأدوات التى يمكن اسـتخدامها فى هذا المجال، وسـوف نركز فى هذا الكتاب على أسلوب تطوير البرمجيات المبنى على الكائنات وهو أسـلوب يلائم المشـاريع كبيـرة الحجم والقابلة للاتسـاع، وسوف نسـتعين ببعض التطبيقات والحالات الدراسـية لتعزيز عملية فهم كيفية استخدام تلك الأساليب، كما سنعرض أيضاً بعض أساليب وأدوات إدارة مشاريع البرمجيات.

هذا، ويحتوى الكتاب على عشـرة فصول فى جزأين: يتناول الجزء الأول المفاهيم الأساسـية لتطوير وإدارة مشاريع البرمجيات والأدوات المسـتخدمة وذلك من خلال الفصـول الأربعة الأولى، حيث نقدم فى الفصل الأول عرضاً لمراحل دورة حياة تطوير

البرمجيـــات مع وصف مختصر لطرق تطوير البرمجيات. ونعرض فى الفصل الثانى أســاليب نمذجة البرمجيات مع التركيز على أســلوب النمذجة المبنى على الكائنات. ونعرض فى الفصل الثالث لبعض أدوات هندسة البرمجيات مثل أدوات إدارة مشاريع البرمجيات وأدوات نمذجة البرمجيات وأدوات تطوير البرمجيات. ونقدم فى الفصل الرابع ببعض التفصيل عمليات وإجراءات إدارة مشـــاريع البرمجيات. ويتناول الجزء الثانى من الكتاب عمليات تطوير مشـــاريع البرمجيات باستخدام الأسلوب المبنى على الكائنـــات. حيث نعرض فى الفصل الخامـــس لكيفية نمذجة متطلبات نظام الأعمال مع الاســـتعانة بحالة دراســـية لتعزيز الفهم. ونقدم فى الفصل السادس بعض الأطر المعمارية المســـتخدمة فى تصميم النظام. ونتناول فى الفصل الســـابع كيفية تصميم قواعد بيانات النظام. كما نقدم فى الفصل الثامن طرق تصميم الأصناف والتفاعلات بينها داخل النظام. وفى الفصل التاســـع نوضح كيفية استخدام لغة البرمجة «جافا» فى تطوير البرمجيات بالإضافة إلى عرض لمراحل اختبار البرمجيات المنتجة. ونقدم فى الفصل العاشـــر عرضاً لطرق تصميم واجهات اســـتخدام النظام مع التركيز على استخدام تقنية الإنترنت لعرض البرمجيات المنتجة على الويب.

ونأمـــل أن يســـاعد هذا الكتاب طـــلاب العلم فى الوطن العربى فـــى مجال تقنية المعلومـــات فى التعرف على كيفية اســـتخدام الأســـاليب والأدوات الحديثة فى تطوير وإدارة البرمجيـــات، وأن يكون إضافة جادة للمكتبة العربية فى هذا المجال، وعلى الله قصد السبيل.

المؤلف: د. عبدالوهاب كامل السماك
sammaka@ipa.edu.sa
sammaka@gmail.com

الجزء الأول

مشاريع البرمجيات

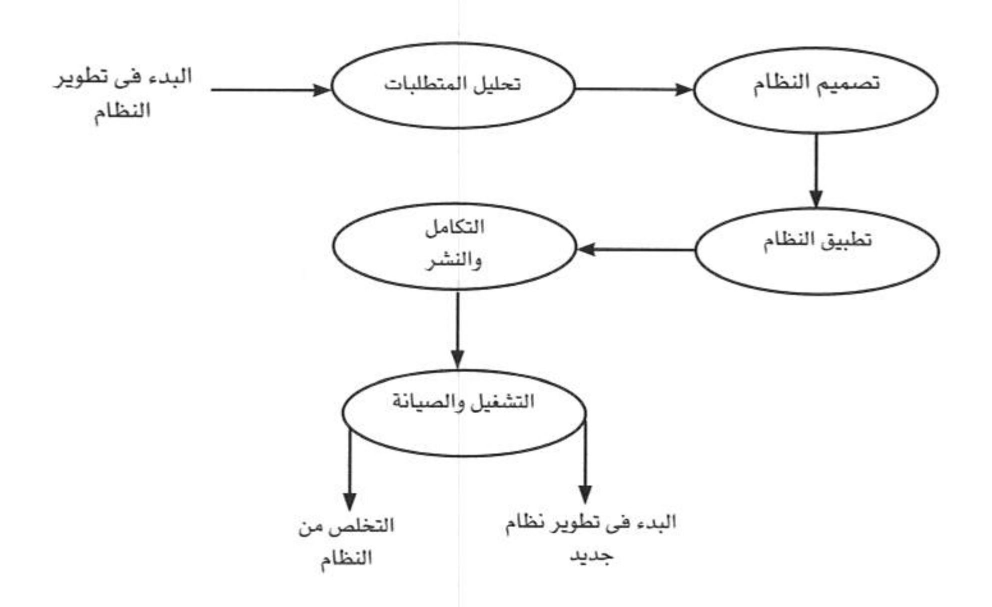
Software Projects

تعتمد معظم بلدان العالم الآن على استخدام الأنظمة المبنية على الحاسب الآلى فى إنشاء وتشغيل وصيانة مشاريع البنية الأساسية الخاصة بها فى القطاعات المختلفة مثل قطاعات الصناعة والزراعة والصحة والتعليم والتجارة والبنوك والخدمات وغيرها. لذلك فإن عملية إنتاج وصيانة البرمجيات بشكل اقتصادى تعتبر ضرورة ملحة للاقتصاد الوطنى والدولى. تركز عمليات هندسة البرمجيات على تطوير وإنتاج برمجيات عالية الجودة بشكل اقتصادى. وقد نشأت فكرة هندسة البرمجيات فى أواخر الستينيات عند دراسة الأزمات والمشكلات التى بدأت تصاحب عملية تطوير البرمجيات فى ظل التقدم المستمر فى تقنيات الحاسب الآلى، وكذلك زيادة حجم التطبيقات المطلوبة واحتوائها على قدر كبير من التعقيدات. ولقد أوضحت تلك الدراسة أن السبب الرئيسى فى تلك الأزمات يرجع إلى ضعف الأساليب المنهجية التى يتم تطوير البرمجيات على أساسها، مما أدى إلى إنتاج البرمجيات فى وقت أكبر من المتوقع وبتكلفة أكبر بكثير من التكلفة المقدرة وبكفاءة تشغيل ضعيفة، بالإضافة إلى صعوبة صيانتها فيما بعد. وقد تطلب ذلك استحداث طرق وأساليب وأدوات جديدة تساهم فى تطوير وإنتاج البرمجيات كبيرة الحجم بكفاءة وجودة عالية وبشكل اقتصادى أيضاً.

ولقد أدى ازدياد الاعتماد على استخدام الطرق الجديدة لهندسة البرمجيات فى بناء النظم، واستحداث تقنيات جديدة خصوصاً فى مجالات أنظمة الاتصالات واندماجها مع أنظمة الحاسبات، إلى زيادة المتطلبات والاحتياجات لأنظمة البرمجيات مما زادها تعقيداً. وعليه فقد أصبح الاستمرار فى عمليات التطوير والتحديث فى أساليب وأدوات هندسة البرمجيات أمراً ضرورياً لإمكانية السيطرة على عمليات تطوير وإنتاج البرمجيات المعقدة بشكل جيد. لأن مشاريع تطوير البرمجيات كبيرة الحجم تعتبر مكلفة وتستغرق وقتاً طويلاً ويقوم على تنفيذها فرق عمل مختلفة وتحتوى على قدر كبير من الخطورة، فإنها تحتاج إلى إدارة جيدة لكى تمكنها من تحقيق أهدافها بشكل دقيق. وسوف نعرض فى هذا الجزء بعض المفاهيم الأساسية لتطوير وإدارة مشاريع البرمجيات والأدوات المستخدمة فيها.

الفصل الأول
دورة حياة تطوير البرمجيات
Software Development Lifecycle

كما أن لكل كائــن حى دورة حياة تبدأ بولادته وتنتهى بوفاته، فإن للبرمجيات دورة حيــاة تصف التغيرات التى تحدث للمنتج البرمجى خلال مراحل تطويره المختلفة منذ نشــأته واستخدامه وصيانته إلى أن تصبح عملية صيانته والتحديث فيه صعبة وتكون جدوى الاســتمرار فى استخدامه ضعيفة ويتم الاســتغناء عنه. ومن المعروف أن تلك التغيرات تحدث بالتدريج فى شــكل مراحل تطوير تتســم بالنمو التدريجى والتحديث والتوســعة المســتمرة، حتى يتم اتخاذ القرار بالاستغناء عنه واســتبداله بنظام آخر. ومــن الجدير بالذكــر أن عملية الإحــلال أو التحويل إلى نظام آخــر بديل تتم أيضاً بالتدريج. يوضح شــكل (١-١) المراحل المختلفة لدورة حياة البرمجيات، والتى سوف يتم استعراضها بالتفصيل لاحقاً فى هذا الفصل.



شكل (١-١) دورة حياة البرمجيات

تتكون دورة حياة البرمجيات من المراحل التالية (Maciaszek, 2005):

١- تحليل المتطلبات Requirements Analysis.

٢- تصميم النظام System Design.

٣- التطبيق Implementation.

٤- التكامل والنشر Integration and Deployment.

٥- التشغيل والصيانة Operation and Maintenance.

يتضح من الشكل (١-١) أنه فى حالة التقديم لمنتج برمجى لمؤسسة ما وبمروره بمراحل التطوير المختلفة واستخدامه فعلياً وازدياد اعتماد المؤسسة عليه فى إنجاز الأعمال وحتى وصوله إلى حالة عدم المقدرة على تحقيق المتطلبات الجديدة للمؤسسة أو أن تكون صيانة النظام كبيرة جداً ومكلفة إلا أنه من الصعب جداً أن تتخذ المؤسسة القرار بالتخلص منه والعودة إلى النظام اليدوى التقليدى لذلك يكون من الأفضل البدء فى تطوير نظام جديد، والتخطيط لاستبدال النظام الحالى بنظام آخر يحقق متطلبات المؤسسة من خلال تطبيق مراحل التطوير مرة أخرى من البداية، وهو ما يسمى بدورة حياة المنتج البرمجى. ومن الضرورى أن تتم عملية الإحلال بالتدريج، أى أنه يتم الاستمرار فى استخدام النظام الحالى فى أثناء مرحلة تطوير النظام الجديد وتثبيته واستخدامه، ويتم استخدام النظامين، القديم والجديد، لفترة حتى يتم التأكد من جودة وفائدة النظام الجديد فى تحقيق متطلبات المؤسسة، وبعد ذلك يتم الاستغناء عن النظام القديم.

١-١ أساسيات هندسة البرمجيات Software Engineering Fundamentals:

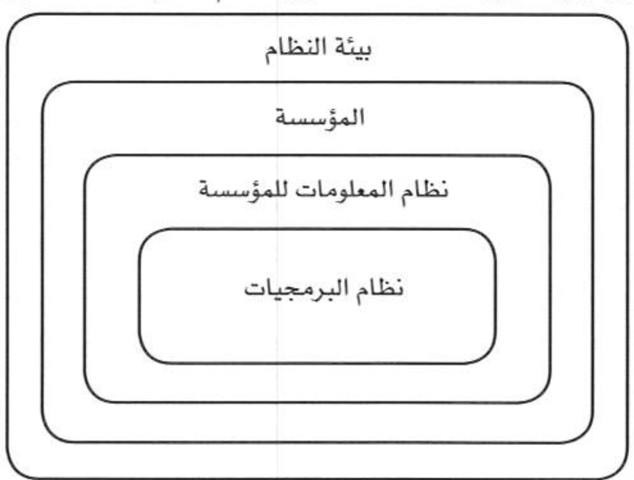
سوف نبدأ الآن بعرض أساسيات هندسة البرمجيات وذلك قبل المضى فى شرح تفاصيل دورة حياة البرمجيات. وسنقوم باستعراض تلك الأساسيات من خلال مجموعة من المفاهيم الأساسية التالية:

- النظام البرمجى Software System.
- العملية البرمجية Software Process.
- هندسة البرمجيات Software Engineering.
- نمذجة البرمجيات Software Modeling.

وسوف نستعرض الآن كل مفهوم بشىء من التفصيل.

١-١-١ النظام البرمجى Software System:

يعتبــر النظام البرمجى جزءاً من نظام أكبر وهو نظام المعلومات الخاص بالمؤسســة Enterprise Information System، وهــو يعتبر أحد مكونات المؤسســة، وهى بدورها تعتبر جزءاً من بيئة نظم الأعمال، كما هو موضح بالشكل (١-٢). وهذا يعنى أن عملية تطوير النظام البرمجى تعتبر أحد أنشطة تطوير نظام المعلومات الخاص بالمؤسسة.



شكل (١-٢) مخطط بيئة نظم الأعمال

تهتم نظم المعلومات بعملية إنتاج وإدارة المعلومات وتقديمها للمســتفيد، مما يمكنه من إنجاز أعماله بكفاءة وجودة أعلى من ذى قبل، وتســاعده على التطوير والاتســاع واتخــاذ القرارات الصحيحة فى الوقت المناســب مما يعود بالنفع على نظام الأعمال.
يحتوى نظام المعلومات على مجموعة من المكونات المترابطة مثل:

- الأفراد.
- البيانات والمعلومات.
- الإجراءات وقواعد العمل.
- البرمجيات.
- الأجهزة.
- الاتصالات.

فمـــن الطبيعى أن يحتوى نظـــام المعلومات على مجموعة مـــن القواعد وإجراءات العمـــل التى يتم تطبيقها علـــى مجموعة من البيانات أو المعلومـــات، وذلك من خلال مجموعة المستخدمين الذين يقومون بتشغيل النظام فعلياً باستخدام أجهزة الحاسبات الآليـــة وملحقاتها وتقوم بتنفيذ مجموعة من البرمجيات، التى قد تحتاج إلى الاتصال بأجهزة أخرى تمدها بالمعلومات فى أثناء التشغيل.

١-١-٢ العملية البرمجية Software Process:

المقصود بالعملية Process عموماً هو الكيفية التى يتم بها إنجاز المهام أو أنشـــطة العمل المطلوبة لإنتاج منتج أو تقديم خدمة. ومن المعروف أيضاً أن البرمجيات Software تســـاهم بشـــكل فعال فى إنجاح نظام الأعمال Business وذلك من خلال الدور الذى تقوم به العمليـــة البرمجية فى إصدار المنتج أو تقديم الخدمة المطلوبة بكفاءة. وعلى هـــذا فإن العملية البرمجية تعتبر جزءاً أساســـياً من نظـــام الأعمال، حيث يتم تقديم هذه الخدمة خلال المســـتويات الإدارية المختلفة مثل مســـتوى التشغيل Operational Management، والمستوى التكتيكى Tactical Management، والمستوى الإستراتيجى Strategic Management، كما هو موضح فى الشكل (١-٣).

إدارة إستراتيجية	نظم المعرفة
إدارة تكتيكية	نظم المعلومات الإدارية
إدارة تشغيلية	نظم معالجة البيانات

شكل (١-٣) العملية البرمجية للمستويات الإدارية المختلفة

تقوم العمليـــة البرمجية غالباً بتقديم الخدمة لأحد المســـتويات الإدارية المذكورة. فعلى مستوى التشـــغيل تقوم العملية البرمجية بمعالجة البيانات الأولية وإنجاز المهام الأساســـية لنظام الأعمال مثل معالجة طلبات العملاء وإصدار الفواتير وغيرها، وهو الجزء الأكبر من نظم الأعمال ويتم الاستعانة فيه بتقنية قواعد البيانات Data Bases.

وعلى المستوى التكتيكى يتم تحليل المعلومات المستخلصة من مستوى التشغيل والاستفادة منها فى توجيه نظام الأعمال مثل تحليل ودراسة طلبات العملاء فى فترة معينة، وكذلك اتجاه المبيعات فى فترة ما واتخاذ القرارات التصحيحية المناسبة لتحسين عائدات نظام الأعمال، ويتم ذلك بالاستعانة بتقنية مخازن البيانات Data Warehouse. وعلى المستوى الإستراتيجى يتم تقنين المعرفة المستخلصة من المستويين السابقين من خلال مجموعة من الحقائق والقواعد التى تصف نظم المعرفة Knowledge Systems لنظام الأعمال، ويتم ذلك بالاستعانة بتقنية قواعد المعرفة Knowledge Bases. وبهذا يتضح الدور الذى تقوم به العملية البرمجية خلال المستويات الإدارية الثلاثة والقيمة التى تقدمها لنظام الأعمال.

١-١-٣ هندسة البرمجيات Software Engineering:

تعتبر هندسة البرمجيات هى أحد أشكال النظم الهندسية التى تهتم بدراسة المبادئ والأسس التى تتحكم فى تطوير وتشغيل وصيانة النظم المعقدة مثل النظم الميكانيكية والنظم الكهربية والنظم الإنشائية وغيرها. ولكن تختلف هندسة البرمجيات عن باقى النظم الهندسية فى عدة أشياء أساسية نذكر منها ما يلى:

المنتج Product:

يكون المنتج غالباً فى النظم الهندسية على شكل مواد Materials سواء كانت صغيرة الحجم مثل الأدوات التى نستخدمها فى حياتنا اليومية والأجهزة الدقيقة، أو كانت كبيرة الحجم مثل المبانى والكبارى ومحطات الكهرباء والسدود وغيرها. أما المنتج فى هندسة البرمجيات فيكون فى شكل برمجيات فقط سواء كانت صغيرة الحجم وقليلة الاستخدام مثل البرمجيات الخاصة بالأفراد التى تساعدهم فى إنجاز أعمالهم بشكل ما، أو كانت كبيرة الحجم معقدة تستخدم على نطاق واسع لعديد من المستخدمين فى أنحاء العالم مثل برامج نظم التشغيل ومترجمات لغات البرمجة ونظم قواعد البيانات وغيرها.

المبادئ Principles:

يتم صياغة المبادئ المكتشفة فى هندسة النظم فى شكل نماذج رياضية أو نظريات مثبتة يتم تطبيقها فى أثناء عملية تطوير المنتجات الهندسية. أما المبادئ المكتشفة

فى هندسة البرمجيات فيتم صياغتها فى عدة أشكال، منها ما يأخذ الشكل الرياضى المنطقى ومنها ما يأخذ شكل المنطق المضبب (غير الواضح) Fuzzy Logic، ومنها ما يأخذ شكل التوجيه لاكتشاف الأشياء Heuristic الذى يتحقق بدون قواعد مثبتة معتمداً على خبرات سابقة.

الإنتاج Production:

بينما يتم إنتاج المنتجات الهندسية من خلال عمليات التصنيع، فإن إنتاج البرمجيات يتم من خلال عمليات التطوير Development. هناك نوعان أساسيان للمنتج البرمجى هما:

برمجيات عامة: هى برمجيات تقوم بتنفيذ الوظائف الأساسية لتشغيل واستخدام الحاسب ويتم استخدامها على نطاق واسع للعديد من المستخدمين مثل نظم التشغيل Operating Systems ومعالجة النصوص Word Processing ونظم إدارة قواعد البيانات Database Management Systems (DBMS) وغيرها. ومن المعروف أنه يتم تطوير هذه الأنظمة من خلال شركات عالمية فى مجال صناعة البرمجيات وهى تقوم بإنتاج آلاف من النسخ المتطابقة للمنتج الواحد بغرض التوزيع.

برمجيات مخصصة: هى نظم برمجيات يتم تطويرها لتلائم متطلبات عميل معين وتعمل فى بيئة تشغيل معينة، مثل نظم الأعمال التجارية والإدارية والتحكم الخاصة بالمؤسسات أو المنشآت الخاصة ذات الطابع التجارى أو الإدارى أو التقنى.

ويتفق الهدف من المنتج البرمجى مع الهدف من المنتج الهندسى فى كونهما يحققان متطلبات وأهداف العميل والمستخدم بكفاءة وجودة معينة وأنهما يتسمان بإمكانية الصيانة. ولكنهما يختلفان فى إمكانية التوسعة. فبينما تتسم البرمجيات بقدرتها على الاتساع Extension والترقية Upgrade حسب التغيرات التى تنشأ فى بيئة العمل، فإن هذه العملية غير ممكنة فى أغلب النظم الهندسية حيث يتم تطوير نظام آخر جديد ومطور بدلاً من النظام المستخدم حالياً.

وعلى هذا فإنه يمكننا تعريف هندسة البرمجيات بأنها أحد مجالات علوم الحاسب الآلى التى تهتم بعملية إنتاج نظم البرمجيات كبيرة الحجم التى تحتوى على قدر من التعقيد مما يتطلب اشتراك فريق عمل أو فرق عمل متخصصة فى أثناء تطويرها. يتكون فريق العمل القائم على تطوير النظام البرمجى من مجموعة أفراد يمتلكون

مجموعة مــن الخبرات والمهارات التى تمكنهم من إنجاز المهــام المطلوبة منهم بكفاءة وجودة عالية، بالإضافة إلــى التفاعل الإيجابى والتكامل فيما بينهم فى أثناء عمليات التطويــر المختلفــة. بالطبع لا تقتصر المهارة المطلوبة لأفــراد فريق العمل على مهارة البرمجة فقط ولكن هناك مهارات أساســية أخــرى مطلوبة لبناء النظام مثل مهارات تحليــل نظام الأعمال وتحديــد المتطلبات وتصميم الحلول وتصميــم قواعد البيانات وتصميم واجهات المستخدم واختبار الوحدات المنتجة واختبار النظام العام ونشر وتوزيع النظام، بالإضافة إلى مهارات تحديد مواصفات الأجهزة المطلوبة وتصميم الشــبكات ومهارات اســتخدام أدوات تطوير وإدارة المشــروع. ولهذا فــإن القائمين على تطوير أنظمة هندســة البرمجيــات يجب أن يتوافر لديهم العديد مــن المهارات التى تؤهلهم للمشــاركة الفعالة فى واحد أو أكثر من أنشــطة تطوير وإدارة مشــاريع البرمجيات. ولذلــك فإن مجال هندســة البرمجيات يعتبر من المجالات الغنيــة بالمعرفة والمهارات والأدوات والتقنيات التى تســتخدم بشــكل جيد من قبل فــرق العمل لإنتاج برمجيات تحقق متطلبات نظم الأعمال فى المجالات المختلفة.

١-١-٤ نمذجة البرمجيات Software Modeling:

تعتمد هندســة البرمجيات على أسلوب النمذجة Modeling باعتباره أسلوباً فعالاً لوصف الأنشــطة التى تتم خــلال مراحل التطوير المختلفة لنظــام البرمجيات، حيث تستخدم النماذج فى تمثيل أو وصف أحد المكونات الوظيفية الفعلية لنظام الأعمال أو جزء منه وذلك بشكل مرئى بسيط، ومن خلال منظور معين يركز على أحد المفاهيم أو الأساليب المستخدمة لتطوير النظم. كما تستخدم النماذج أيضاً بوصفها وسيلة فعالة للتفاعل والتواصل بين أفراد فريق العمل بالمشــروع وكذلك مع المســتفيد. يســتخدم مهندس البرمجيات أســلوب النماذج التوضيحية خلال المراحــل المختلفة لدورة حياة تطوير النظم البرمجية مثل:

نموذج المتطلبات Requirements Model:

هو نموذج يصف متطلبات المســتفيد الوظيفية باســتخدام إحــدى طرق النمذجة المنهجيــة. كمثال على ذلك نســتخدم مخطــط تدفق البيانات DFD فى الأســلوب الهيكلى، ونســتخدم مخططــات حالات الاســتخدام Use Cases والأصناف Classes والتتابع Sequence والتفاعل Interaction وغيرها فى الأســلوب المبنى على الكائنات (الفصل الثانى).

النموذج المعمارى Architectural Model:

هو نمـــوذج يصف معمارية النظام البرمجى من خـــلال تمثيل النظم الفرعية التى يتكـــون منها النظام والتفاعلات فيما بينها مثل مخططات الحزم Packages والمكونات Components من خلال مجموعة من الأطر المعمارية المختلفة (الفصل السادس).

نموذج التصميم التفصيلى Detailed Design Model:

هو نمـــوذج يصف المكونات التفصيليـــة لكل من البرمجيـــات Software والأجهزة Hardware المطلوبـــة لتطبيق الحلول المقترحة للنظام البرمجى مثل تصميم الأصناف والتفاعلات (الفصل الثامن) وتصميم مخطط النشر (الفصل الثانى).

ومن الممكن أن تتم تجزئة أى نموذج من تلك النماذج إلى عدة نماذج أخرى تفصيلية مثل نماذج وصف منطق العمليات ونماذج قواعد البيانات ونماذج واجهات المســـتخدم وغيرها.

تعتمـــد نوعية النماذج المســـتخدمة على المنهجية المســـتخدمة فـــى تطوير النظام البرمجى. وهناك مفهومان أساسيان يمكن استخدامهما لتطوير النظم هما:

المفهوم الوظيفى Functional Paradigm:

يقوم هذا المفهوم على أســـاس تجزئة النظـــام إلى وحدات فرعية صغيرة حتى يتم الســـيطرة عليها وتطويرها بســـهولة، وبعد ذلك يتم تجميع الوحدات الفرعية لتكوين النظـــام العام. ويســـتخدم هذا المفهـــوم نماذج تدفق البيانـــات Data Flow Models لوصف عمليات النظام.

مفهوم الكائنات الموجهة Object-Oriented Paradigm:

يقوم هذا المفهوم على أســـاس تجزئة النظام إلى مكونات وظيفية صغيرة تســـمى الكائنات Objects مـــع توصيف لطرق الاتصال بينها كما هو الحال فى الواقع الفعلى لنظام الأعمال. يستخدم هذا المفهوم نمذجة الأصناف Classes Modeling ونمذجة حالات الاســـتخدام Use-Case Modeling وغيرها من النماذج. وســـوف يتم عرض هذه النماذج بالتفصيل فى الفصل القادم.

١-٢ مراحل دورة حياة تطوير البرمجيات Software Development Lifecycle Phases:

تعتبر دورة حياة البرمجيات وصفاً موجزاً لإستراتيجية تطوير البرمجيات. فهى تصف مراحل وخطوات وأنشطة وطرق تطوير مشروع البرمجيات، بالإضافة إلى تمثيل المخرجات المتوقعة للمشروع خلال المراحل المختلفة. تتضمن دورة حياة تطوير البرمجيات المراحل التالية:

- تحليل المتطلبات Requirement Analysis.
- تصميم النظام System Design.
- التطبيق Implementation.
- التكامل والنشر Integration and Deployment.
- التشغيل والصيانة Operation and Maintenance.

تتضمن كل مرحلة من هذه المراحل العديد من الأنشطة، ولكل مرحلة مدخلاتها ومخرجاتها وتأثيرها فى جودة المنتج النهائى. وسوف نستعرض الآن كل مرحلة بشىء من التفصيل.

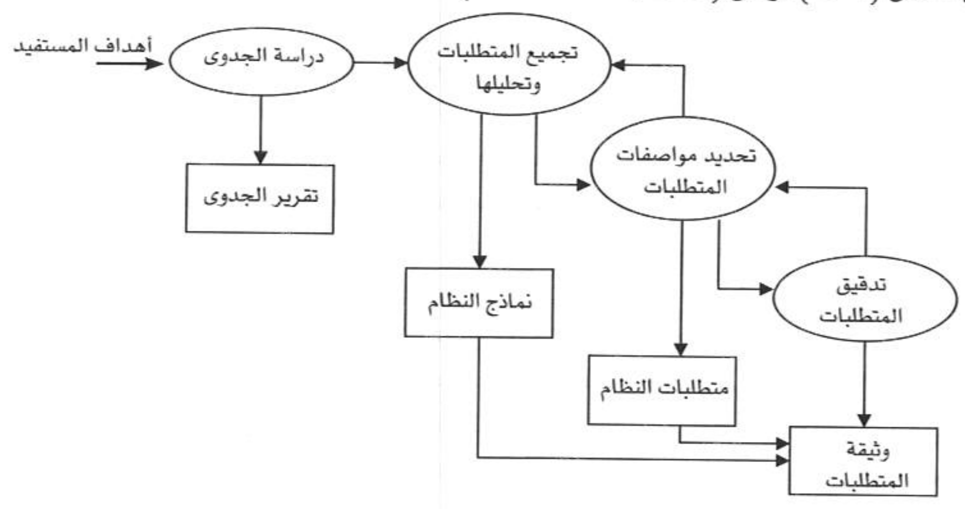
١-٢-١ تحليل المتطلبات Requirements Analysis:

تعتبر عملية دراسة وتحليل متطلبات مستخدمى النظام هى أول خطوة فى عملية تطوير نظم البرمجيات. تحتوى هذه العملية على مجموعة من الأنشطة التى تهدف إلى تحديد متطلبات واحتياجات مستخدمى النظام. المقصود بمتطلبات المستخدمين هو مجموعة الخدمات التى يجب أن يقدمها النظام للمستخدمين، مع وصف لمجموعة القيود التى سوف تحكم عملية تنفيذ الخدمات. ولذلك تعتبر عملية تحديد المتطلبات من التحديات المهمة التى تواجه عملية تطوير البرمجيات وخصوصاً فى الحالات التى يكون فيها المستخدمون غير متفهمين لاحتياجاتهم بدقة ووضوح. فأحياناً يبالغ المستخدم فى وصف المتطلبات أو يحدد احتياجات تتعارض مع احتياجات أخرى لمستخدم آخر داخل النظام. وأحياناً أخرى لا يعطى المستخدم هذه العملية الأهمية المطلوبة فتكون النتيجة إنتاج برمجيات لا تفى بالخدمات المطلوبة وتحتاج إلى تعديل وإضافة الكثير مما يتطلب وقتاً ومجهوداً وميزانية إضافية عن المقدرة بالمشروع. من ثم فإن القائمين على تطوير النظام يعانون بشدة من مسألة عدم فهم مستخدمى النظام

للمتطلبــات بدقة ووضــوح مما يتطلب تدخل الأفراد القائمــين على تحليل المتطلبات واستخدام خبراتهم ومهاراتهم المكتسبة من أنظمة أخرى مشابهة لمساعدة المستخدمين فى تحديد متطلباتهم. تؤدى عملية تحديد المتطلبات إلى تعريف شكل النظام المطلوب وتحديــد الوظائف والمهام التى يجب أن يقوم بهــا النظام بكفاءة. هناك مجموعة من الطرق والأساليب التى يتم اتباعها فى أثناء عملية تحديد متطلبات النظام مثل:

- المقابلات مع المستخدمين وخبراء فى المجال.
- تصميم استبانات للمستخدمين.
- ملاحظة المستخدمين فى أثناء أدائهم للمهام.
- دراسة وثائق ومستندات النظام الحالى.
- دراسة أنظمة مشابهة للتعلم منها واكتساب معارف خاصة بالمجال.
- إعداد نماذج تجريبية لتجربة الحلول المقترحة والتأكد من جدواها.
- عقد جلسات عمل بمشاركة فريق التطوير وباقى المشاركين فى المشروع.

تتضمــن عمليــة تحديد متطلبات النظام أربعة أنشــطة رئيســية، كما هو موضح بالشكل (١-٤)، وهى (Sommerville, 2004):



شكل (١ - ٤) عملية تحديد المتطلبات

١- دراسة الجدوى Feasibility Study:

تهدف دراسة الجدوى إلى تحديد مدى إمكانية تحقيق أهداف المستفيد باستخدام التقنيات المتاحة وفى حدود الميزانية المقدرة للمشروع. تنتهى الدراسة بتوصيات تساعد المستفيد على اتخاذ القرار بالمضى قدماً فى تنفيذ المشروع أو التوقف لعدم جدواه.

٢- تجميع المتطلبات وتحليلها Requirements Collection & Analysis:

يتم فى هذا النشاط تجميع معلومات عن متطلبات النظام من خلال ملاحظة النظام الحالى ومناقشة المستخدمين ودراسة النماذج والوثائق الخاصة بالنظام. وبعد ذلك يتم إعداد النماذج التى تصف بعض متطلبات النظام الوظيفية والهيكلية، والتى تساعد فى عملية فهم النظام ومداولته مع باقى المشاركين فى المشروع بشكل أفضل.

٣- تحديد مواصفات المتطلبات Determining Requirements Specification:

يتم فى هذا النشاط تحويل المعلومات المجمعة فى النشاط السابق إلى وثيقة تصف جميع متطلبات النظام. هناك نوعان من المتطلبات هما:

- متطلبات وظيفية Functional Requirements: عبارة عن مجموعة الوظائف والعمليات التى يجب أن يقدمها نظام الأعمال فى شكل مجموعة من الخدمات المقدمة للمستفيد من النظام.
- متطلبات غير وظيفية Nonfunctional Requirements: عبارة عن مجموعة من القيود التى تحكم تنفيذ المتطلبات الوظيفية لنظام الأعمال مثل بيئة التشغيل وقيود الأمن والسلامة وغيرها.

٤- تدقيق المتطلبات Requirements Validation:

يتم فى هذا النشاط التحقق من واقعية Realism واتساق Consistency واكتمال Completeness المتطلبات.

يتم استخدام أسلوب النمذجة Modeling لتوصيف المتطلبات التى تم تحديدها، وذلك باستخدام إحدى طرق النمذجة القياسية مثل لغة النمذجة الموحدة UML التى يتم من خلالها وصف متطلبات النظام فى شكل مجموعة من المخططات Diagrams، بالإضافة إلى مجموعة من النماذج النصية Textual Models. ومن المعروف أن النظم المعقدة لا يمكن فهمها بشكل جيد ومتكامل باستخدام وجهة نظر أو من منظور

واحد Viewpoint، ولكن يتم تحليل المتطلبات من خلال وجهات نظر مختلفة تساعد على تعزيز عملية فهم وتوصيف متطلبات النظام بشكل جيد. وفى حالة استخدام إحدى أدوات هندسة البرمجيات المدعمة بالحاسب CASE فإنه يمكننا تخزين جميع النماذج التخطيطية والنصية التى تصف متطلبات النظام فى المستودع الرئيسى للنظام Repository الذى يتم الاستفادة منه فى جميع مراحل تطوير النظام، بالإضافة إلى إنتاج وثائق النظام System Documentations بشكل آلى ودقيق. وتجدر الإشارة هنا إلى أنه فى حالة عدم استخدام تلك الأدوات المساعدة، فإنه يجب على مدير المشروع أن يركز على متابعة فرق التطوير بالمشروع، وضرورة استكمال توثيق العمليات التى يقومون بتنفيذها، ومن أولها وأهمها تحديد متطلبات النظام. يمكننا أيضاً الاستعانة ببعض القوالب Templates الخاصة بتوثيق المتطلبات المقدمة من الشركات العالمية المتخصصة أو تصميم قالب توثيق خاص بالمؤسسة. يحدد القالب غالباً الهيكل العام للوثيقة، بالإضافة إلى مجموعة من الإرشادات عن كيفية تعبئتها. تحتوى الوثيقة على النماذج والنصوص التى تصف مجموعة الخدمات التى يجب أن يقدمها النظام، بالإضافة إلى وصف لمجموعة القيود التى يتضمنها النظام فى أثناء تأدية تلك الخدمات مثل الأمن والأداء وظروف التشغيل وبعض القيود القانونية. وللتحقق من جودة مخرجات عملية تحديد المتطلبات، يتم مراجعة واختبار تلك المخرجات من قبل مجموعة متخصصة فى المجال تابعة لإدارة ضمان جودة البرمجيات بالمؤسسة Software Quality Assurance. تتكون هذه المجموعة من خبراء متخصصين فى اختبار البرمجيات وهم من خارج فريق التطوير للمشروع، وهم بذلك يصبحون مشاركين فى مسؤولية ضمان جودة البرمجيات المنتجة.

١-٢-٢ تصميم النظام System Design:

تهتم عملية تصميم البرمجيات بالأنشطة الخاصة بوصف كل من هيكل البرمجيات، هيكل البيانات، وواجهات استخدام النظام. أحياناً لا يتم وصف خوارزميات العمليات بشكل مفصل فى أثناء مرحلة التصميم وذلك لإتاحة قدر أكبر من الحرية للمبرمجين فى أثناء مرحلة التطبيق. وفى حين كانت عملية التحليل غير مقيدة بأى قيود تخص بيئة التشغيل الأساسية التى سوف يعمل النظام من خلالها سواء كانت تخص العتاد Hardware أو تخص البرمجيات Software، فإن عملية التصميم تأخذ فى الاعتبار عدة اعتبارات وقيود خاصة ببيئة التشغيل التى سيعمل النظام من خلالها. فى الواقع العملى لا يوجد خط فاصل واضح بين مرحلتى التحليل والتصميم وذلك لسببين أساسيين. السبب الأول هو أن الطرق الحديثة لتطوير البرمجيات تعتمد على أسلوب

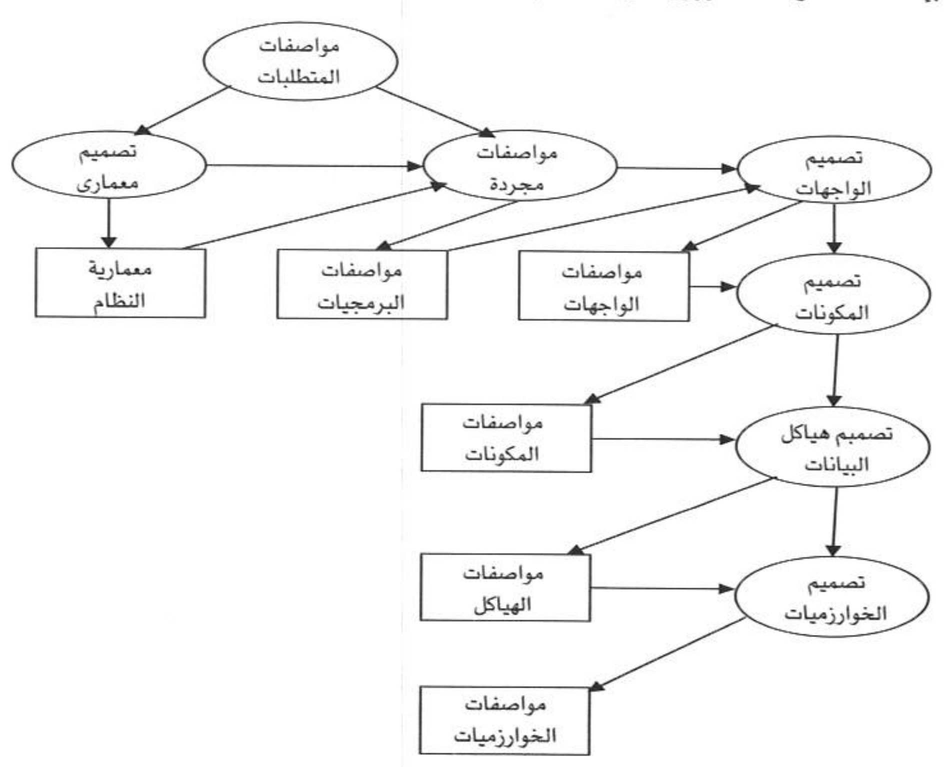
التكرار مع الزيادة Iterative with Increment وهو يتسم بإمكانية ظهور عدة إصدارات مختلفة للنظام تحت التطوير، ويكون أحد الإصدارات فى مرحلة التحليل وإصدار آخر فى مرحلة التصميم وإصدار ثالث فى مرحلة التطبيق، وهكذا. والسبب الثانى يرجع لطبيعة أسلوب النمذجة المستخدم، فمثلاً يتسم أسلوب لغة النمذجة الموحدة UML بأنه يتم استخدام بعض النماذج لكل من عمليتى التحليل والتصميم. وعلى هذا فإنه يمكننا القول إن الانتقال من عملية التحليل إلى عملية التصميم يتم بأسلوب الإسهاب Elaboration وذلك بإضافة مواصفات تفصيلية إلى النموذج المستخدم فى أثناء عملية التحليل وليس بأسلوب التحويل من نموذج إلى نموذج آخر. يسمى هذا النوع باسم «التصميم التفصيلى Detailed Design». هناك نوع آخر من التصميم يسمى «التصميم المعمارى Architectural Design» وهو يعنى بإعداد الإطار المعمارى للنظام وهيكل المكونات للتصميم التفصيلى وأطر الاتصالات الداخلية بين المكونات. ونستطيع أن نوجز القول بأن الهدف الأساسى للتصميم المعمارى هو الوصول إلى نظام مفهوم قابل للصيانة والاتساع. تتكون عملية التصميم من مجموعة الأنشطة التالية، كما هو موضح بشكل (١- ٥) (Sommerville, 2004).

- تصميم معمارى Architectural Design.
- مواصفات مجردة Abstract Specifications.
- تصميم الواجهات Interface Design.
- تصميم المكونات Component Design.
- تصميم هياكل البيانات Data Structure Design.
- تصميم الخوارزميات Algorithm Design.

١-٢-٣ التطبيق Implementation:

على الرغم من أن المقصود بتطبيق النظام هو برمجته، إلا أن عملية برمجة النظام لا تعنى مكوث مجموعة من المبرمجين فى مكان واحد والقيام بكتابة البرامج باستخدام إحدى لغات البرمجة. فكما ذكرنا من قبل أن عملية التصميم لا تقوم بتوصيف جميع الخوارزميات التفصيلية، ولكن يتم تركها لمرحلة البرمجة، حيث يقوم المبرمج بتنفيذ العمليات حسب رؤيته والأسلوب البرمجى الذى يفضله ويحسن استخدامه، مع الأخذ فى الاعتبار أن يتم ذلك طبقاً للمواصفات المذكورة فى وثائق التحليل والتصميم. ومن

النادر اليوم أن يقوم المبرمج بكتابة أكواد جميع البرامج، ولكنه يستعين بمكتبة البرامج واستخدام بعض المكونات البرمجية الجاهزة، مما يوفر الوقت والمجهود. وهذا يتطلب أن يكون المبرمج على دراية جيدة بمحتويات مكتبة البرامج وكيفية اســتخدامها بشكل جيد. يقوم المبرمج أيضاً بعمليتى الهندسة الأمامية والهندسة العكسية فى أثناء عملية التطبيــق وخصوصاً إذا كان مســتخدماً لإحدى أدوات هندســة البرمجيات المدعمة بالحاســب Computer Aided Software Engineering (CASE) وفــى الحقيقة، إن هذه الأدوات تساعد كثيراً فى عمليات إنتاج قواعد البيانات وتوليد البرامج الأساسية طبقاً لمواصفات التصميم التفصيلية التى تم إدخالها، وبعد ذلك يقوم المبرمج بالتدخل بإضافة الأكواد الضرورية لربط المكونات البرمجية المنتجة وتعديلها حسب الحاجة.



شكل (١-٥) نموذج عام لعمليات التصميم

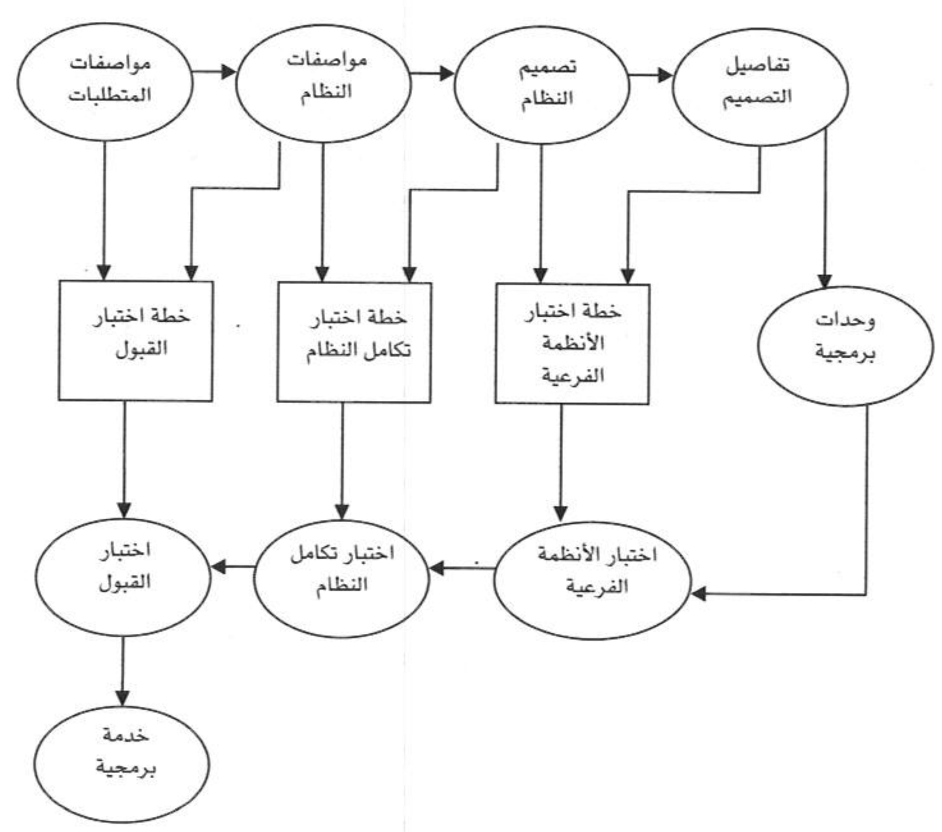
وبعد الانتهاء من عملية اختبار المنتجات البرمجية يقوم المبرمج بعملية إعادة هندسة البرمجيات المنتجة عكسياً لتعديل مواصفات التصميم المدخلة من قبل، وذلك لضمان

التوافق بين مواصفات التصميم والبرمجيات المنتجة، وقد يتطلب ذلك مشاركة كل من المبرمجين والمصممين فى مراجعة هذه العملية وإجراء التعديلات الضرورية على كل من الأكواد ومواصفات التصميم. من المعروف أن مرحلة التطبيق تستغرق الوقت الأكبر خلال مراحل تطوير النظام، والسبب فى ذلك لا يرجع إلى حجم الأكواد المكتوبة فحسب إنما يرجع أيضاً إلى كثرة الأخطاء والإضافات والتعديلات التى تتم على الأكواد فى أثناء عمليات المراجعة والاختبار للبرمجيات المنتجة. أحياناً يقوم المبرمج بنفسه بعمليات المراجعة والاختبار لمنتجاته، ولكن من الأفضل أن يقوم بها فريق آخر، إذ يتضمن مجموعة من محللى النظام والمستخدمين ويقومون بفحص واختبار البرامج المنتجة باستخدام بيانات حقيقية ومعدة مسبقاً لاختبار البرامج ومقارنة النتائج بما هو متوقع.

١-٢-٤ التكامل والنشر Integration and Deployment:

تتضمن عملية التكامل أنشطة تجميع الأنظمة الفرعية Sub-Systems التى تم تطويرها واختبارها من قبل. من الممكن أن يتم تجميع جميع الأنظمة الفرعية معاً فى آن واحد لتكوين النظام العام، ولكن من الأفضل أن تتم عملية التكامل بالتدريج وبشكل متزايد مما يضمن نجاحها فى وقت أقل وبمجهود أقل من الطريقة الأولى. فمن عيوب الطريقة الأولى أنها تتطلب أن يتم الانتهاء من تطوير واختبار جميع الأنشطة الفرعية أولاً، وهو طلب غير عملى وخاصة بالنسبة للأنظمة الكبيرة. بالإضافة إلى أن عملية تكامل أو تجميع عدد كبير من الأنظمة الفرعية فى آن واحد تحتاج إلى مجهود كبير لاكتشاف الأخطاء التى تحدث فى أثناء عملية التكامل، وتستغرق عملية التغلب عليها مجهوداً ووقتاً كبيراً. فى حين أن إجراء عملية التكامل بالتدريج يوفر المجهود والوقت؛ لأن الأخطاء التى تحدث فى أثناء تكامل نظام فرعى معين مع نظام حالى تكون غالباً من النظام الجديد أو نتيجة التفاعلات بين النظامين ويمكن اكتشافها والتغلب عليها بسهولة. وبعد نجاح المرحلة الأولى لعملية التكامل يتم إضافة نظام فرعى آخر للنظام الحالى وتكرار هذه العملية حتى يتم الحصول على النظام العام. ولضمان إتمام عملية التكامل للنظام بنجاح، يتم اختبار التفاعلات والاتصالات بين الأنظمة الفرعية، ومن ثم اختبار أداء النظام العام وهو ما يسمى باختبار التكامل Integration Test. يوضح شكل (١-٦) مراحل اختبار النظام فى أثناء عمليات التطوير والتسليم. والجدير بالذكر أن طريقة تنفيذ عملية التكامل

تتطابق بدرجة كبيرة مع التصميم المعمارى للنظام، إذ إن هذا التصميم يوضح مكونات النظــام والاعتمادية بينها . ومــن الممكن أن تتم عملية التكامل بين مكونات النظام من أعلى إلى أســفل Top-Down أو من أسفل إلى أعلى Bottom-Up أو بأسلوب وسط وهو خليط بين الأسلوبين يسمى Middle-Out.



شكل (١-٦) مراحل اختبار النظام

تهتم عملية النشــر Deployment بأنشــطة تســليم النظام المطور للعميل وتثبيته فــى بيئة العمل الطبيعية للبدء فى التشــغيل الفعلى. تتم هــذه العملية بالتدريج كما هــو الحال فى عملية التكامل، فيتم نشــر البرمجيات المنتجة على مراحل فى شــكل إصدارات مختلفة. فيتم تثبيت الإصدار الأول للنظام واختباره من قبل المســتخدمين فى بيئة العمل الطبيعية، ويطلق على هذا النشاط اسم اختبار ألفا Alpha Test. يتم تحديث هذا الإصدار عند ظهور أى أخطاء أو ملاحظات من المســتخدمين تســتدعى

التعديـل فيه. وبعد ذلك يتـم إضافة مكونات أخرى للنظام إلى الإصدار الحالى ويتم اختبارهـا أيضاً فى بيئة العمـل الطبيعية وبذلك يتم إنتاج إصدار ثـانٍ للنظام. يتم تكرار تلك العملية حتى نصل إلى إنتاج الإصدار الأخير للنظام وهو يحتوى على جميع مكونات النظام العام. وفى النهاية يتم اختبار النظام العام من قبل المستخدمين للتأكد مـن تحقيقه لجميع الأهداف والمتطلبات المذكـورة فى مواصفات الاحتياجات للنظام، وهو ما يسـمى باختبار القبول Acceptance Test. تتضمن عملية نشـر النظام عدة أنشطة أخرى مثل تدريب مستخدمى النظام وهى يمكن أن تبدأ قبل عملية نشر النظام فى الحالات التى تتطلب تدريب المسـتخدمين على اسـتخدام أدوات أو تقنيات أخرى ضرورية لتشـغيل النظام نفسـه. ومن أنشطة عملية النشـر أيضاً عملية إنتاج دليل استخدام النظام للمستفيدين (User Documentation (Guide.

١-٢-٥ التشغيل والصيانة Operation and Maintenance:

يبـدأ المنتـج البرمجى المنفذ الذى تم نشـره عند العميل بتنفيـذ إجراءات العمل اليومية أو ما يسـمى بعمليات التشـغيل اليومية وذلك فى ظـل وجود النظام القديم. يتم تشـغيل النظامين (القديم والجديد) على التوازى لبعض الوقت. وبعد التأكد من كفاءة تشـغيل النظام الجديد فى إنجاز الأعمال اليومية يتم خروج النظام القديم من التشـغيل واسـتمرار تشغيل النظام الجديد. ومع بداية التشـغيل الفعلى للنظام تبدأ فعليـاً عمليات الصيانة Maintenance للنظام. هنـاك ثلاثة أحوال يتم فيها صيانة النظام تحت التشغيل، هى:

- إصلاح العيوب والأخطاء التى يتم اكتشافها فى أثناء عمليات التشغيل.
- إجراء تعديلات فى البرمجيات المنتجة لتلائم التغييرات التى تحدث فى بيئة نظام الأعمال.
- إضافـة إمكانيات جديدة للبرمجيات المنتجة لتحسـين جودتها والارتقاء بها لصالح نظام الأعمال أو نتيجة لاتساع النظام نفسه.

من المعروف أن المؤسسـات الكبيرة ترصد ميزانية سـنوية لأعمال الصيانة لنظام الأعمال تساهم فى تحديث وتطوير النظام بشكل مستمر مما يحمى نظام الأعمال من التقادم أو الوصول إلى مرحلة التوقيف نتيجة انعدام الفائدة من استخدامه.

١-٣ طرق تطوير البرمجيات Software Development Methods:

كمـــا ذكرنـــا من قبل فـــإن العملية البرمجيـــة تحتوى على مجموعة من الأنشـــطة المترابطة التى تؤدى إلى إنتاج منتج برمجى معين، وهى عملية معقدة تختلف تفاصيلها من مؤسسة لأخرى ومن مشروع لمشروع آخر، إذ إنها تتأثر بعدة عوامل منها:

- خبرات ومهارات الأشخاص القائمين على تنفيذ الأنشطة.
- مجال التطبيق Application Domain لنظام الأعمال.
- التغيـــرات الخارجيـــة التى تؤثر فى بيئة نظام الأعمال مثل التغيرات السياســـية أو الاقتصادية أو الاجتماعية أو التقنية.
- التغيرات الداخلية لنظام الأعمال مثل التغيير فى الإدارة أو ظروف العمل أو الموقف المالى للمؤسسة أو فى أفراد فريق العمل.
- حجم مشروع نظام الأعمال.

ومع ذلك فإن معظم مشـــاريع نظم الأعمال تشترك فى احتوائها على مجموعة من الأنشـــطة الأساسية التى تستخدم فى تطوير نظم الأعمال وتؤدى إلى تسليم منتجات برمجية تفى بمتطلبات المســـتفيد من النظام. تتضمن الأنشـــطة الأساســـية عمليات تحديـــد وتحليل مواصفات الاحتياجـــات، وتصميم الحلول المقترحـــة للنظام واختيار الأنسب منها، وتطبيق الحلول التى تم اختيارها، واختبار البرمجيات المنتجة وتكاملها، ثم عمليات نشر البرمجيات المنتجة والتحول إلى النظام الجديد، ثم صيانة النظام بعد بدأ التشغيل الفعلى.

إلا أن طريقة تطوير مشـــاريع البرمجيات تختلف حســـب طبيعة المشروع وظروفه والقيـــود التى تحكمه. هناك العديـــد من الطرق والمنهجيات التى تســـتخدم لتطوير مشـــاريع البرمجيات. تمثـــل كل طريقة العمليـــة البرمجية من منظـــور معين ويقدم معلومات حول العملية البرمجية. نعرض هنا لبعض الأساليب الشائعة الاستخدام فى تطوير مشاريع البرمجيات مثل (Maciaszek, 2005):

- أسلوب الشلال Waterfall .
- أسلوب التكرار مع الزيادة Iterative with Increment .
- هندسة البرمجيات المبنية على المكونات Component-Based Software Engineering .

- التشييد المبنى على النموذج (Model Driven Architecture (MDA.
- أسلوب النماذج التجريبية Prototyping.

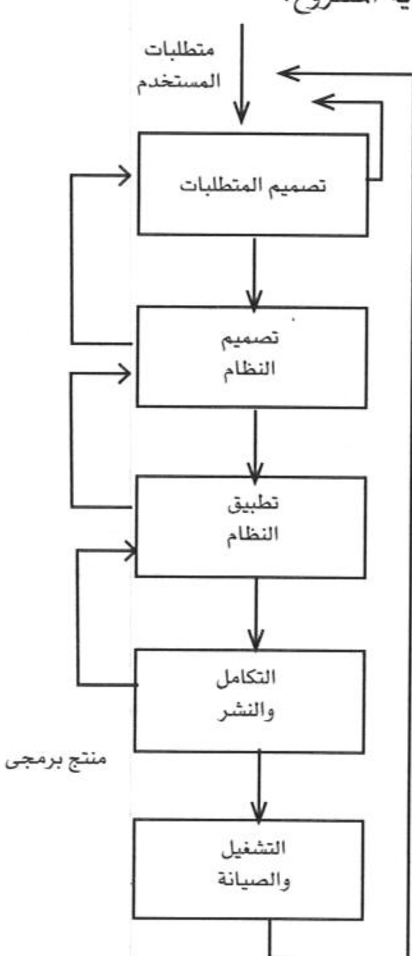
والجدير بالذكر هنا أنه يمكننا الاستفادة من استخدام أكثر من أسلوب فى تطوير المشــروع بخاصة المشــاريع الضخمة. فعلى ســبيل المثال هناك أسلوب تطوير جديد يســمى بالعملية العقلانية الموحــدة (Rational Unified Process (RUP وهو يجمع عناصر عديدة من أغلب الأســاليب السابقة الذكر. والآن سوف نقوم باستعراض كل أسلوب بشىء من التفصيل.

١-٣-١ طريقة الشلال Waterfall Method:

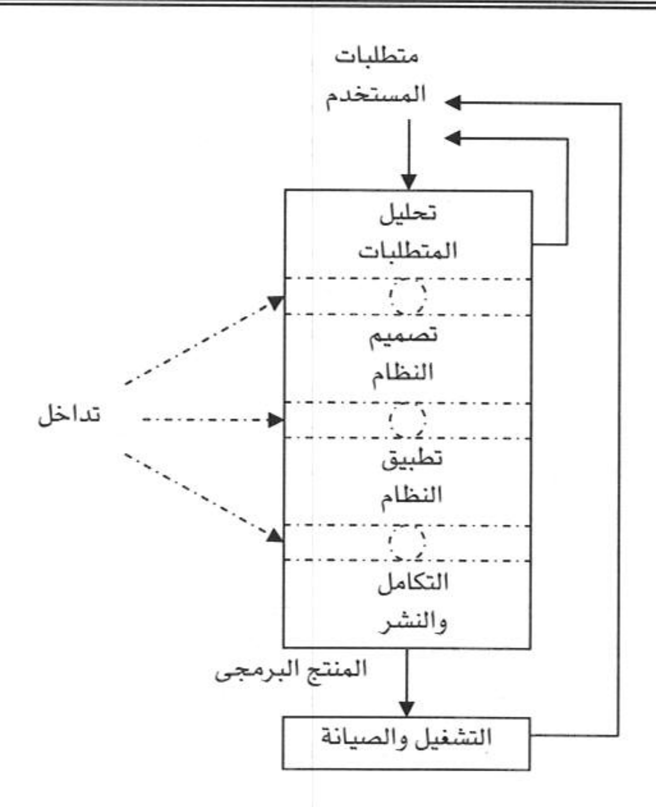
تعتبر طريقة الشــلال من الطرق التقليدية التى شــاع اســتخدامها لتطوير النظم فى الســبعينيات وقد حققت نجاحاً كبيراً فى ذلك الوقت، حيث كانت تناســب حجم وطبيعــة النظم وقتها. أما الآن فقد قل اســتخدامها نظراً للتطورات التى حدثت فى التقنيات المســتخدمة ومن ثم فــى المتطلبات. يوضح شــكل (١-٧) المراحل المختلفة لطريقة الشلال مع إمكانية الارتداد (التغذية العكسية) Feedback بين المراحل. تتسم طريقة الشلال بأسلوب المراحل المتتالية التى لا يتم فيها الانتقال من مرحلة حالية إلى مرحلة تالية إلا بعد الانتهاء من تنفيذ جميع أنشطة المرحلة الحالية، مع إمكانية العودة من أى مرحلة إلى المرحلة السابقة لها لإجراء بعض التعديلات، ويمكن أن يستمر هذا الارتداد حتى الوصول إلى المرحلة الأولى. هناك عدة أشكال يمكن أن تأخذها طريقة الشلال، فمنها ما يسمح بالتداخل Overlap بين المراحل المتتالية كما هو موضح بشكل (١-٨)، وفيها يمكن لأى مرحلة أن تبدأ قبل اكتمال أنشطة المرحلة السابقة لها.

تتميز طريقة الشلال بأنها تقوم بإنتاج التوثيق الخاص بكل مرحلة فى نهاية المرحلة، مما يضمن اكتمال عملية توثيق النظام. وفى المقابل فإن طريقة الشلال تعانى من عدة مشاكل فى أثناء تطوير النظم الكبيرة التى تتسم بكثرة التغيرات فيها وتحتوى على قدر من التعقيد. وتكمن غالباً المشــكلات من أســلوب التقسيم غير المرن الذى تستخدمه طريقة الشلال فى تجزئة عمليات تطوير النظام إلى عدة مراحل متعاقبة، مما يستلزم ضــرورة اكتمال ووضوح جميع متطلبات العميل من النظام فى المرحلة الأولى وصعوبة تغييرها فيما بعد. فإذا اســتحدثت متطلبات جديدة فى مراحل متأخرة من المشروع، فإن ذلك ســوف يتطلب البدء فى دورة حياة تطوير جديدة للنظام. وإذا حدث تعديل

فــى المتطلبات فــى أثناء عمليات التطويــر، فإنه يمكن إجراؤها عــن طريق الارتداد للمراحل الســابقة، ولكن ذلك يـؤدى إلى تعطل فرق العمل فـــى المراحل التالية نظراً لاعتمادهــا على المراحل الســابقة لهــا. وعلى هذا يمكننا القول إن طريقة الشــلال تناسب النظم الصغيرة أو النظم التى تتسم بوضوح واستقرار فى الأهداف والمتطلبات وضمان اكتمالها فى بداية المشروع.



شكل (١-٧) طريقة الشلال



شكل (١-٨) طريقة الشلال مع التداخل

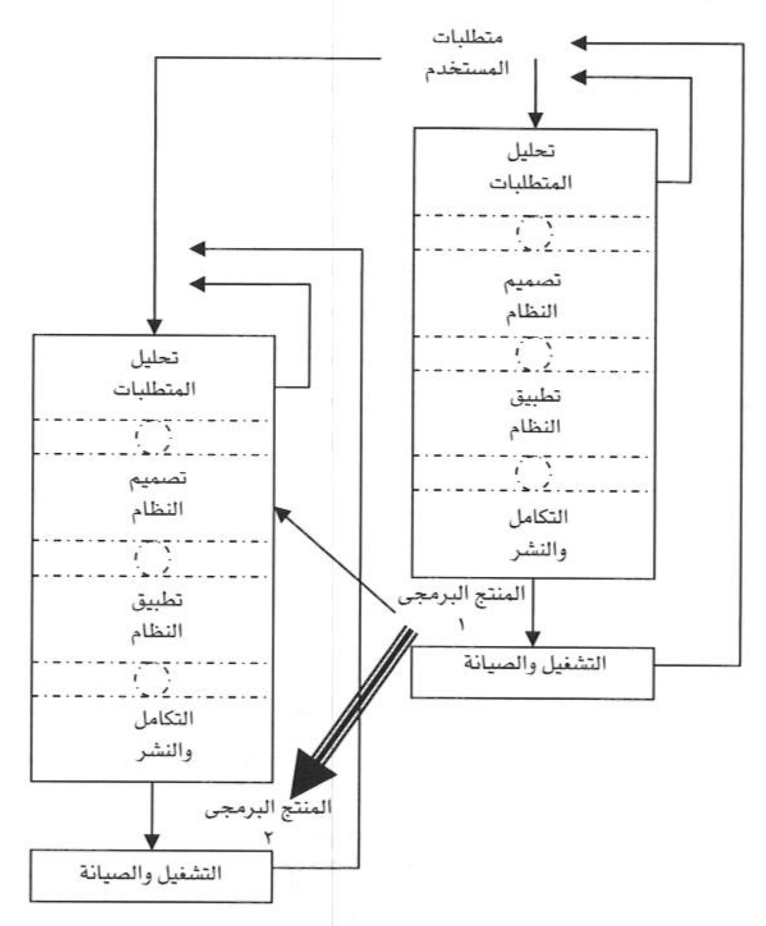
١-٣-٢ طريقة التكرار مع الزيادة Iterative with Increment:

يقصد بالأسلوب التكرارى فى تطوير البرمجيات أنه يتم تكرار تنفيذ مجموعة من العمليـــات البرمجية مع إضافة مكونات جديدة فى كل مرة وذلك بهدف إثراء أو تنمية المنتج البرمجى. يلائم هذا الأسلوب تماماً النظم الكبيرة والمعقدة التى تتعرض لكثير من التغيرات فى بيئة العمل أو التى لا تكون متطلباتها كاملة فى بداية عملية التطوير. ففى تلك الأنظمة يمكننا بالاتفــاق مع العميل تحديد مجموعة الخدمات أو الوظائف المطلوبة وتحديد أســبقية تنفيذ أولوية كل خدمة أو وظيفة حســب أهميتها بالنســبة للعميــل. بعد ذلك يتم تحديد مواصفات الاحتياجات للخدمات ذات الأولوية القصوى أولاً والبدء فى تطويرها وتطبيق جميع عمليات التطوير عليها حتى يتم الحصول على النســخة الأولية للمنتج البرمجى، ويتم اســتخدامها. بعد ذلــك نقوم بإضافة خدمة

جديدة أو أكثر ونكرر الإجراءات نفسها بدءًا من تحديد مواصفات الاحتياجات وباقى عمليـات التطوير وانتهاء بالحصول على النسـخة المطورة الثانية من المنتج البرمجى. وهكـذا حتى ننتهى من إنجاز جميع الخدمـات المطلوبة من النظام. وهنا نلاحظ أنه قد تم إنتاج النظام العام بالتدريج فى شـكل نسـخ مطورة على مراحل تكرارية وليس مـن مرة واحدة كما هو الحال فى طريقة الشـلال. ونلاحـظ أيضاً أنه قد تم تكرار تنفيذ مراحل التطوير كاملة لإنتاج كل نسخة مطورة. وفى حالات أخرى يتعرض نظام الأعمـال لكثير من التغيرات سـواء فى الوظائف أو فى ظـروف العمل أو فى التقنية المسـتخدمة، فإنها تحتاج إلى التعديل فى مواصفـات الاحتياجات ومن ثم تعديل فى التصميم والتطبيق، ومن ثم إصدار نسـخة مطورة للنسـخة الحالية للمنتج البرمجى. وعلى هذا يتم تعريف أسـلوب التطوير بالتكرار مع الزيادة بأنه الأسـلوب الذى يؤدى إلى إصدار نسخ مطورة، سواء بالتحسين أو بالإضافة، للمنتج البرمجى وذلك فى نهاية كل محاولة. يوضح شـكل (١-٩) أسـلوب التطوير بالتكرار مع الزيادة الذى يتضمن مراحل طريقة الشلال فى كل محاولة تكرارية. ويتضح من الشكل أنه بعد الانتهاء من عملية التشغيل وفى أثناء عملية الصيانة تبدأ عملية التكرار بإضافة احتياجات جديدة والاستمرار فى تنفيذ باقى المراحل فى دورة حياة التطوير، وهكذا.

يوفر استخدام أسلوب التطوير بالتكرار مع الزيادة عدة مميزات مهمة مثل:

- عـدم احتياج العميل للانتظار طويلاً حتى يتـم الانتهاء من تطوير النظام العام مرة واحدة، ولكنه يسـتطيع استخدام النسـخة الأولية فى تنفيذ الاحتياجات الضرورية للنظام مبكراً.
- اكتساب العميل خبرة جيدة فى أثناء استخدام النسخة الأولية للنظام مما يمكنه من تحديد باقى الاحتياجات والمتطلبات بشكل دقيق فى النسخ المطورة التالية.
- تكرار عملية اختبار الخدمات والوظائف الأساسية للنظام فى كل مرة يتم فيها اختبار النسخة الجديدة للنظام مما يزيد من جودة المنتج ومن ثم جودة النظام العام.
- يـؤدى ذلك كله إلى التخفيف من المخاطر التـى يمكن أن يواجهها النظام وتؤثر فيه ويمكن أن تؤدى إلى فشـله أو انهيـاره. ويتم هذا التخفيف نتيجة للتغلب التدريجى على المشـكلات التى تنشـأ فى كل محاولة وعدم تركها لتتراكم وتتفاقم فى تأثيرها فى النظام العام.



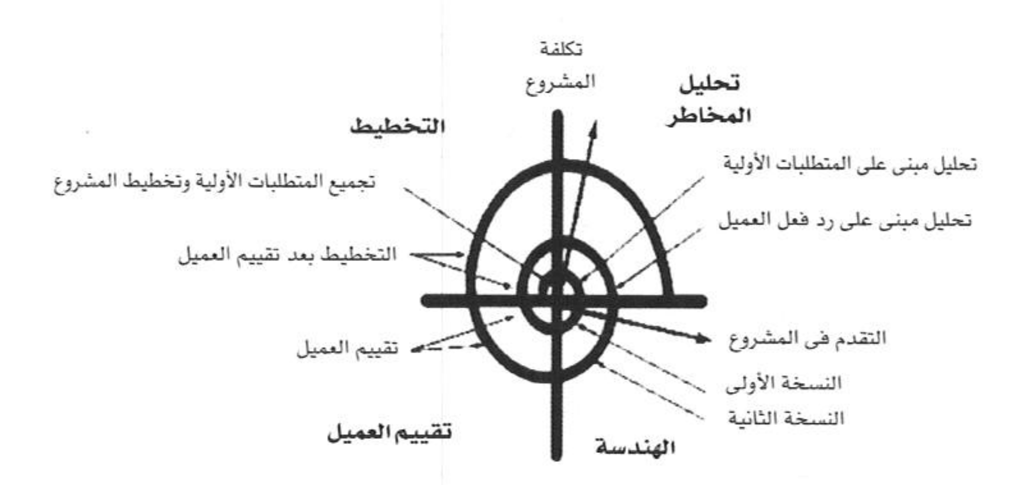
شكل (١-٩) أسلوب التكرار مع الزيادة

هنـــاك طريقتان تنفذان هذا الأســـلوب فى تطوير النظم، الأولى تســـمى «النموذج الحلزونى Spiral Model» وقد ظهرت فى أواخر الثمانينيات، والثانية تسمى «العملية العقلانية الموحدة RUP» والمقدمة من شركة IBM فى عام ٢٠٠٣.

١-٣-٢-١ النموذج الحلزونى Spiral Model:

يتم تمثيل العملية البرمجية فى هذا النموذج بشكل حلزونى Spiral بدلاً من تمثيلها فى شـــكل مجموعة من الأنشـــطة المتتاليـــة. كل حلقة Loop فى هـــذا النموذج تمثل مرحلـــة فـــى العملية البرمجية مثل تحديـــد المتطلبات أو التصميم وغيرها. ينقســـم

النمـــوذج الحلزونى إلـــى أربعة قطاعات Sectors، كما هو مبين فى الشـــكل (١-١٠)، وهـــى قطاع التخطيـــط Planning، قطاع تحليل المخاطـــر Risk Analysis، والقطاع الهندســـى Engineering، وقطاع تقييم العميل Customer Evaluation. تبدأ دائماً الحلقـــة الأولى للنموذج من قطاع التخطيط وهى تمثل عملية تجميع المتطلبات الأولية ومن ثم تخطيط المشروع. بعد ذلك يدخل المشروع فى مرحلة (قطاع) تحليل المخاطر التـــى تبدأ بتحليل التكلفة/المنفعة وتحليل الفرص المتاحة للمســـاعدة فى عملية اتخاذ القرار بالاســـتمرار والانتقال إلى مرحلة التطوير (القطاع الهندســـى) أو التوقف عن تنفيذ المشـــروع لعدم جدواه. يهتم القطاع الهندسى بعمليات التطوير Development التى تنتج النسخة الأولية للنظام البرمجى التى تخضع بعد ذلك لتقييم العميل (قطاع تقييـــم العميل). وبعد ذلك تبدأ الحلقة الثانية فى النموذج الحلزونى لإنتاج النســـخة الجديدة المطورة للنظام.



شكل (١-١٠) النموذج الحلزونى لتطوير النظم

نلاحـــظ أن هـــذا النموذج قد تعامل مـــع عمليات التطوير من خـــلال قطاع واحد وهو القطاع الهندســـى. ونلاحظ أيضـــاً التركيز على عملية تحليل المخاطر وتكرارها مع كل حلقة مما يســـاعد على الاكتشـــاف المبكر لأى مخاطر يمكن أن تواجه المشروع ويمكنه التغلب عليها بطريقة أســـرع وأسهل من تراكمها ومن ثم يخفف من آثارها فى

المشـــروع. ونلاحظ أيضاً تكرار عملية التخطيط فى بداية كل حلقة، وعملية التقييم للمنتـــج البرمجى من قبـــل العميل فى نهاية كل حلقة، مما يحســـن مـــن جودة المنتج البرمجى. والملاحظة الأخيرة هى أن كل جزء من الحلقة داخل القطاع الهندسى تمثل محاولة تكرارية Iteration للمنتج البرمجى وهى تمثل مرحلة من مراحل التطوير حتى الوصول إلى المنتج النهائى.

١-٣-٢-٢ العملية العقلانية الموحدة Rational Unified Process (RUP):

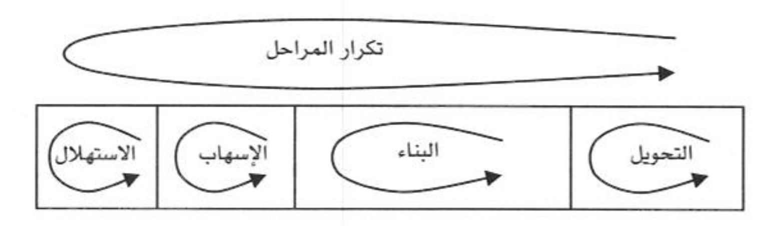
تعتبـــر العملية العقلانية الموحدة أحد نمـــاذج العملية الحديثة لتطوير النظم والتى تم اقتباســـها من أسلوب لغة النمذجة الموحدة UML. وهى تعتبر مثالاً جيداً للعملية المجمعة Hybrid Process التى تحتوى على مجموعة من العناصر المجمعة من الطرق السابقة. فهى تحتوى على عدة مراحل مثل طريقة الشلال Waterfall، ومع ذلك فهى تدعم أســـلوب التكرار مع الزيـــادة Iterative with Increment، بالإضافة إلى كونها تتســـم بواقعية أكثر فـــى أثناء عمليات تحديد المواصفـــات والتصميم للنظام. يمكننا وصف العملية العقلانية الموحدة من خلال ثلاثة منظورات هى:

١- منظور ديناميكى Dynamic يصف مراحل تطوير النظام حسب الوقت.

٢- منظور ساكن Static يصف أنشطة العملية.

٣- منظور عملى يقترح مجموعة من الممارســـات الجيدة المكتســـبة من خبرات سابقة لتطبيقها خلال عمليات التطوير للنظام الحالى.

يصف المنظور الديناميكى مراحل تطوير النظام حسب الوقت وليس حسب الأنشطة، وهو يتكون من أربع مراحل أساسية كما هو مبين فى الشكل (١-١١):



شكل (١-١١) المنظور الديناميكى للعملية العقلانية الموحدة RUP

١- مرحلة الاستهلال Inception Phase:

الهدف من هذه المرحلة هو تأسـيس حالة نظام الأعمـال وذلك من خلال تحديد جميع الكائنات الخارجية External Entities التى تتفاعل مع النظام، مع تعريف لتلك التفاعـلات. يتم اسـتخدام تلك المعلومات لتحديد الدور الـذى يقوم به النظام تحت التطوير ضمن مشروع نظام الأعمال العام.

٢- مرحلة الإسهاب Elaboration Phase:

الهدف من هذه المرحلة هو فهم مجال التطبيق وتأسيس الإطار العام للنظام وإعداد خطة المشـروع وتحديد النقاط التى تحتوى على بعض المخاطر بالمشروع. ويتوقع فى نهايـة تلك المرحلة أن يتم إنجاز نموذج مواصفات المتطلبات للنظام والوصف المعمارى للنظام، بالإضافة إلى إعداد خطة تطوير المشروع.

٣- مرحلة البناء Construction Phase:

تهتم تلك المرحلة بالأنشـطة الخاصة بتصميم وبرمجة واختبار النظام. يتم إنجاز أجزاء من تلك الأنشـطة على التـوازى أو التداخل، ثم يتم التكامل بينها فى أثناء تلك المرحلة. وفى نهاية المرحلة يمكننا الحصول على نظام برمجى جاهز للتشغيل ومرفقاً معه نماذج توثيق النظام المطور.

٤- مرحلة التحول Transition Phase:

الهدف من تلك المرحلة هو الانتقال من مرحلة التطوير إلى مرحلة التشغيل فى بيئة التشغيل الطبيعية. وفى نهاية هذه المرحلة يكون لدينا نظام برمجى موثق يعمل بصورة صحيحة فى بيئة التشغيل الفعلية.

تدعم العملية العقلانية الموحدة RUP أسلوب التكرار مع الزيادة على مستويين هما:

داخلى: يتم التطوير بالتدريج باستخدام أسلوب الزيادة داخل كل مرحلة.

خارجى: يتم اسـتخدام أسـلوب الحلقة بـين جميع المراحل المتتاليـة فى العملية العقلانية.

يركز المنظور السـاكن للعملية العقلانية على الأنشـطة التى يتم تنفيذها فى أثناء عملية التطوير. ويتم توصيف أنشـطة المشروع من خلال مجموعة من تدفقات العمل

Workflows، وهى عبارة عن ستة تدفقات عمل أساسية وثلاثة تدفقات عمل مساندة، كما هو مبين فى الشـكل (١- ١٢). والجدير بالذكر هنا أن العملية العقلانية الموحدة RUP قد تم تقديمها بالاشـتراك مع لغة النمذجة الموحدة UML، ولهذا فإننا سـوف نجد أن وصف تدفقات العمل يكون موجهاً نحو نماذج UML المصاحبة لها.

تدفق العمل	الوصف
نمذجة نظام الأعمال	توصيف عمليات نظام الأعمال باستخدام نموذج حالات الاستخدام.
تحليل متطلبات النظام	توصيف متطلبات مستخدمى النظام من خلال تفاعل مجموعة الفاعلين مع حالات استخدام النظام.
التصميم	إنشاء نماذج التصميم المعمارى والمكونات وتتابع الكائنات للنظام.
التطبيق	تطبيق مكونات وإجـراءات نظام الأعمال من خلال مجموعة من الوحدات البرمجية المتفاعلة.
الاختبار	اختبار كل وحدة برمجية منتجة وكذلك اختبار تكامل الوحدات داخل النظام.
النشر	إصدار المنتج البرمجى وتوزيعه وتثبيته عند العميل.
إدارة التشكيل والتغيير	دعم المنتج البرمجى وإدارة عمليات التغيير التى تطرأ عليه.
إدارة المشروع	إدارة عمليات تطوير مشروع نظام الأعمال.
بيئة المشروع	توفير مجموعة الأدوات المساعدة التى يحتاج إليها فريق العمل بالمشروع.

شكل (١-١٢) تدفقات العمل الساكنة فى العملية العقلانية الموحدة RUP

ويصف المنظـور العملى للعملية العقلانية الموحدة مجموعة جيدة من الممارسـات العملية لهندسـة البرمجيات التى ينصح بتطبيقها فى تطوير النظم. نذكر هنا بعض الممارسات المهمة التى ينصح باستخدامها.

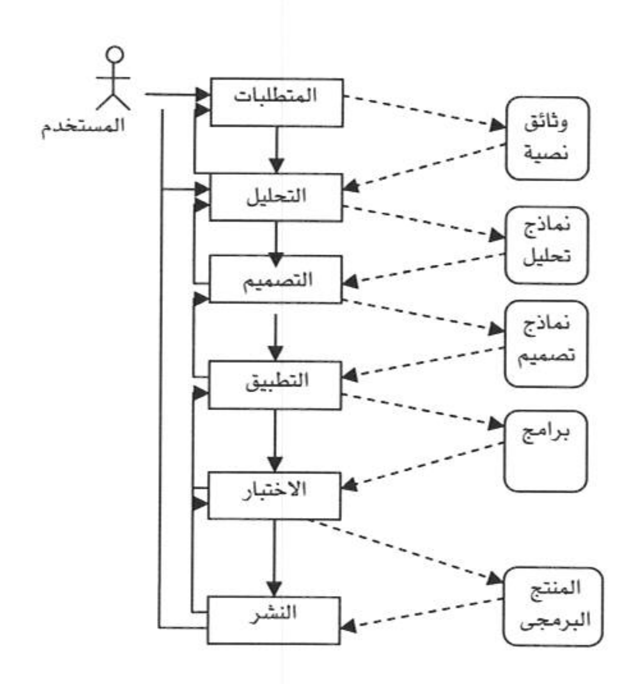
- يجـب تطوير النظام البرمجـى بالتدريج والزيادة، وذلك بناء على الأولويات التى تم تحديدها من قبل العميل، حيث نبدأ فى تطوير واختبار وتسـليم المهام ذات الأولوية القصوى، ثم نضيف إليها مهام أخرى تليها فى الأولوية، وهكذا.
- يجب توثيق جميع متطلبات العميل بشكل دقيق، ومتابعة التغييرات التى تحدث فيها، وتحليل تأثير تلك التغييرات فى النظام قبل قبول تنفيذها.

- مـــن الأفضل أن يتم بناء الهيــكل المعمارى للنظام من المكونات المعدة مســبقاً بقدر الإمكان.
- يفضـــل اســـتخدام النمـــاذج المرئيـــة Visual Models لتمثيل المنظـــورات المختلفة للنظام.
- يجــب التحقق من جــودة البرمجيات المنتجة ومطابقتهـــا لمقاييس الجودة الخاصة بالمؤسسة.
- يجب التحكم فى التغيرات التى تحدث للبرمجيات من خلال إجراءات وأدوات إدارة التغيير.

١-٣-٢-٣ التشييد المبنى على النموذج (Model Driven Architecture (MDA:

قدمـــت مجموعـــة إدارة الكائنـــات (Object Management Group (OMG هـــذا الأسلوب فى محاولة للانتقال بلغة النمذجة الموحدة UML من مرحلة المواصفات الرسمية Formal Specifications إلى مرحلة المواصفات التنفيذية Executable Specifications، أو مرحلـــة إنتـــاج البرامـــج (Programs Generation Maciaszek, 2005). تقوم هذه الفكرة على مبدأ الاســـتفادة بالمعايير القياســـية التى أسستها مجموعة OMG وكذلك الاســـتفادة من التقنيات الحديثة فى تحويل نماذج المواصفات UML Models إلى أكواد قابلة للتنفيذ Executable Codes. يعتبر هذا الأسلوب نوعاً من أنواع النماذج التحويلية Transformational Models التـــى تقوم بتطوير النظام من خلال مجموعة متتالية من التحويلات للنماذج الرســـمية، التى تصف متطلبات النظـــام، إلى مجموعة من البرامج التنفيذيـــة، ويتم ذلك مروراً بمرحلة التصميم التفصيلية. يتم تدقيق كل خطوة تحويلية للتأكد من صحة مخرجاتها وأنها تمثل تحويلاً حقيقياً للمدخلات.

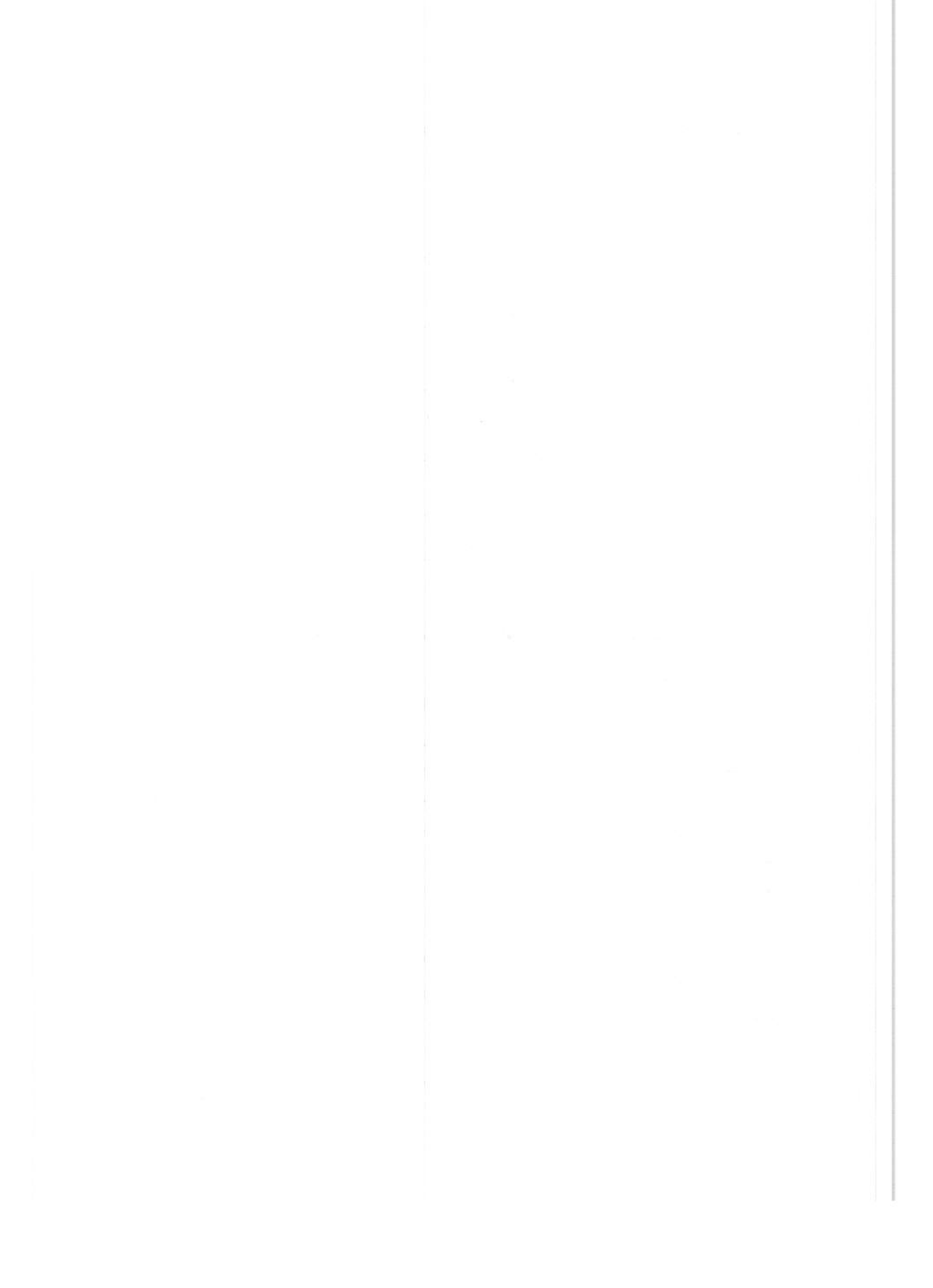
يعتبر هذا الأســـلوب مدخلاً حقيقياً لميكنة عمليات التحويـــل من المواصفات إلى إنتاج البرامـــج، مـــع الأخذ فى الاعتبار أن هنـــاك بعض التحويلات يجب أن تتـــم يدوياً. توجد الآن مجموعة من أدوات هندســـة البرمجيات المدعمة بالحاســـب CASE هى تقوم بتنفيذ تلك التحويلات. تقوم هذه الأدوات بإنشـــاء وتخزين النماذج الرسمية التى تصف مكونات النظام، وبعد ذلك تقوم بتحويلها تدريجياً من خلال مجموعة من النماذج التحويلية. وعلى هذا فإن هذه الأدوات تدعم أسلوب التطوير المتدرج مع الزيادة. يوضح شكل (١-١٣) دورة حياة تطوير النظم باستخدام أسلوب التشييد المبنى على النموذج.



شكل (١-١٣) تطوير النظم باستخدام أسلوب التشييد المبنى على النموذج

يفصل هذا الأسلوب بين الشكل الرسمى للمواصفات وبين الشكل غير الرسمى للمتطلبات، وهو بذلك يقوم باستبعاد الشكل غير الرسمى للمتطلبات من عمليات التحويل، ويبدأ بنماذج المواصفات الرسمية التى يمكن قراءتها آلياً عن طريق الحاسب الآلى. تعتبر هذه النماذج غير مرتبطة ببيئة تشغيل معينة سواء من ناحية العتاد Hardware أو من ناحية البرمجيات Software. وبعد بدء عمليات التحويل تنشأ نماذج التصميم التفصيلية التى تعتمد على بيئة التشغيل. وإذا كان النظام المطور يعمل من خلال بيئات تشغيل متعددة، فإنه يتم توليد نماذج تفصيلية خاصة بكل بيئة تشغيل، وبعد ذلك يتم الربط بينها باستخدام مجموعات ربط تسمى «جسور التشغيل الداخلى Interoperability Bridges».

جدير بالذكر أن هذا الأسلوب للتطوير المبنى على النموذج يعطى الأمل فى إمكانية تطبيق تقنية التطوير بالمكونات التى يمكن من خلالها تطوير النظام بالكامل باستخدام المكونات الجاهزة أو المعدة مسبقاً. تكون هذه المكونات فى شكل نماذج أو برامج يتم إعادة استخدامها لإنتاج العديد من البرمجيات (Pressman, 2001).



الفصل الثانى
نمذجة البرمجيات
Software Modeling

تتطلب عملية تطوير مشاريع البرمجيات قدراً كبيراً من التفاعل والاتصال بين أفراد فرق العمل القائمين على تطوير المشروع فيما بينهم من جهة، وبينهم وبين مستخدمى النظام من جهة أخرى. ولذلك فإنه من الضرورى أن يتم استخدام لغة أو وسيلة محددة لتسهيل عملية الاتصال والتفاعل بين جميع الأفراد المساهمين فى المشروع. يجب أن تكون اللغة المستخدمة لغة عامة ومفهومة لكل المشاركين فى المشروع، وأن تكون لغة قياسية، ومن الأفضل أن تكون ملائمة لعدد كبير من عمليات تطوير المشروع، وذلك لضمان اتساق وانسياب عملية التنقل من مرحلة إلى مرحلة أخرى فى أثناء عمليات التطوير باستخدام الأسلوب نفسه تقريباً.

يعتبر أسلوب النمذجة Modeling من الوسائل المناسبة لتوصيف عمليات تطوير البرمجيات واستخدامه بوصفه وسيلة للتفاهم والاتصال بين فرق العمل وذلك لاعتماده على النماذج المرئية بشكل أساسى لوصف المفاهيم والعمليات للنظام البرمجى. ويمكن أن ترفق مجموعة من النماذج النصية مع النماذج المرئية لوصف محتويات أو تفاصيل النماذج المرئية، بالإضافة إلى أنها تقوم بإضافة معلومات أخرى تعزز عملية فهم نظام الأعمال. ويمكننا القول إن عملية تحديد متطلبات النظام، التى تتم فى بداية عملية التطوير، وكذلك عملية توثيق البرامج المنتجة فى نهاية عملية التطوير، تتم عادة فى شكل نماذج نصية. وفى المقابل فإن المراحل المتوسطة فى أثناء عمليات تطوير النظام، والتحليل والتصميم، تتم عادة باستخدام النماذج المرئية. وعلى هذا فإن عمليات توصيف البرمجيات تتكون فى الغالب من مجموعة من النماذج المرئية Visual Models مصحوبة ببعض النماذج النصية Text Models، وذلك بهدف وصف عمليات التطوير المختلفة للنظام من عدة منظورات مختلفة Viewpoints. يحتوى النموذج غالباً على مخطط Diagram أو أكثر وأى معلومات أخرى توضيحية. يمثل المخطط وصفاً لإحدى وظائف أو مكونات النظام من وجهة نظر معينة وذلك بشكل مرئى باستخدام الرموز والرسومات التخطيطية. يجب تخزين تلك النماذج المرئية والنصية فى المستودع العام للنظام Repository ويتم استخدامه فيما بعد من جميع أفراد فريق التطوير للمشروع. هناك العديد من الأدوات التى تدعم هذه العملية يطلق

عليها اسـم «أدوات هندسـة البرمجيات المدعمة بالحاسـب CASE Tools". سوف نسـتعرض الآن فى هذا الفصل، بشـكل سريع، أسـلوبين فقط من أساليب النمذجة الشـائعة الاستخدام فى تطوير النظم. الأول هو أسـلوب قديم مازال يستخدم حتى الآن وهو «النمذجة الهيكلية Structured Modeling" وهو يناسـب الأنظمة التقليدية التى تم إنتاجها فى السـابق وتسـتخدم أسلوب البرمجة الهيكلية. والثانى هو أسلوب حديث يسـمى «لغة النمذجة الموحدة (Unified Modeling Language (UML" ويتم استخدامه الآن بشكل واسـع وهو يلائم النظم الكبيرة والمعقدة التى تستخدم أسلوب البرمجة الموجهة بالكائنات.

٢-١ النمذجة الهيكلية Structured Modeling:

لقد ظهر أسلوب النمذجة الهيكلية فى أواخر السبعينيات مواكباً لأسلوب البرمجة الهيكلية Structured Programming. تتسـم النمذجة الهيكلية بأسـلوب التصميم الهرمى من أعلى إلى أسـفل Top-Down وهو أسلوب يقوم بتقسيم أو تجزئة النظام إلى مجموعة من الوظائف الفرعية المتفاعلة. تتم عملية التجزئة على عدة مسـتويات من حيث الإيجاز والتفصيل. فيكون الإيجاز فى المستوى الأعلى Top وتكون التفاصيل فى المستويات التالية إلى أسفل Down. يستخدم هذا الأسلوب فى نمذجة متطلبات النظام واسـتخدامها لتعزيز عملية الفهم للنظام والتفاعـل بين أفراد فريق التطوير. تعتمـد النمذجة الهيكلية علـى توصيف عمليات النظـام باعتبارها مدخلاً لتوصيف المهام الوظيفية للنظام. لقد أثبت هذا الأسـلوب نجاحاً كبيراً فى الفترة السابقة فى أثناء اسـتخدامه فى تحليل الأنظمة الكبيرة، وهو يتميز باستخدام رموز وتخطيطات وتدوينات قياسـية، بالإضافة إلى أنه يضمن إنجاز عملية توثيق النظام بشكل قياسى يساهم فى فهم مكونات ومهام النظام بشكل جيد لجميع الأفراد القائمين على تطوير النظـام وصيانته فيما بعد. يعانى أسـلوب النمذجـة الهيكلية من عدة نقاط ضعف، أهمها عـدم المقدرة على توصيف المتطلبـات غير الوظيفية للنظـام، وعدم ملاءمته لأسـلوب التطوير المبنى علـى الكائنـات Object-Orientation. هناك بعض أدوات هندسـة البرمجيات المدعمة بالحاسب CASE التى تقوم بدعم عمليات نمذجة النظم باستخدام أسلوب النمذجة الهيكلية سيتم عرضها فى الفصل الثالث.

تقدم النمذجة الهيكلية مجموعة من النماذج المرئية التى تستخدم فى أثناء عمليات تحليل وتصميم الأنظمة. يمكننا تصنيف تلك النماذج فى شكل مجموعتين رئيسيتين،

تمثل المجموعة الأولى مجموعة النماذج التى تستخدم فى نمذجة العمليات Process Modeling، وتمثل المجموعة الثانية النماذج التى تستخدم فى نمذجة البيانات Data Modeling، وسيتم وصفهما من خلال النماذج التالية:

- نماذج مخططات تدفق البيانات DFDs.
- نموذج مخطط العلاقة بين الكائنات ERD.

وسوف نستعرضها الآن بشىء من التفصيل.

٢-١-١ مخططات تدفق البيانات (DFDs) Data Flow Diagrams:

تعتبر مخططات تدفق البيانات DFDs هى أحد أهم النماذج المستخدمة فى الفترة السابقة لتوصيف المتطلبات الوظيفية لنظم الأعمال. يعتمد أسلوب إنشاء مخطط تدفق البيانات على مفهوم التفكيك الوظيفى Functional Decomposition. يوضح شكل (٢-١) مفهوم التفكيك الوظيفى من خلال تعريف العمليات الوظيفية للنظام فى شكل هرمى أو تدريجى من أعلى إلى أسفل Top-Down. يبدأ تعريف النظام من أعلى بمخطط السياق Context Diagram الذى يصف النظام بشكل عام وتفاعله مع البيئة المحيطة به. وبعد ذلك يتم وصف النظام من الداخل بالتدريج إلى أسفل فى عدة مستويات تفصيلية للعمليات الوظيفية للنظام. تبدأ هذه المستويات بالمستوى «صفر» الذى يصف العمليات الأساسية للنظام ويطلق عليه اسم «مخطط المنظور العام Level 0 Overview Diagram". يمكننا وصف تفاصيل كل عملية أساسية فى المستوى «صفر» من خلال مخططات تفصيلية فى المستوى التالى «واحد» ويطلق عليها اسم «مخططات تفصيلية Level 1 Details Diagrams". تأخذ كل عملية تفصيلية رقم مسلسل ينطلق من رقم العملية الأساسية لها مثل «عملية ١-١» و«عملية ١-٢» وهكذا. وإذا كانت إحدى العمليات التفصيلية فى المستوى «واحد» تحتاج إلى تفاصيل أكثر، يتم وصفها فى شكل مخطط تفصيلى فى المستوى التالى «اثنين» وتأخذ كل عملية تفصيلية رقماً ينطلق من رقم العملية الأم لها مثل «عملية ١-١-٢» وهكذا لباقى العمليات التفصيلية فى جميع المستويات التالية.

العناصر الأساسية لمخططات تدفق البيانات:

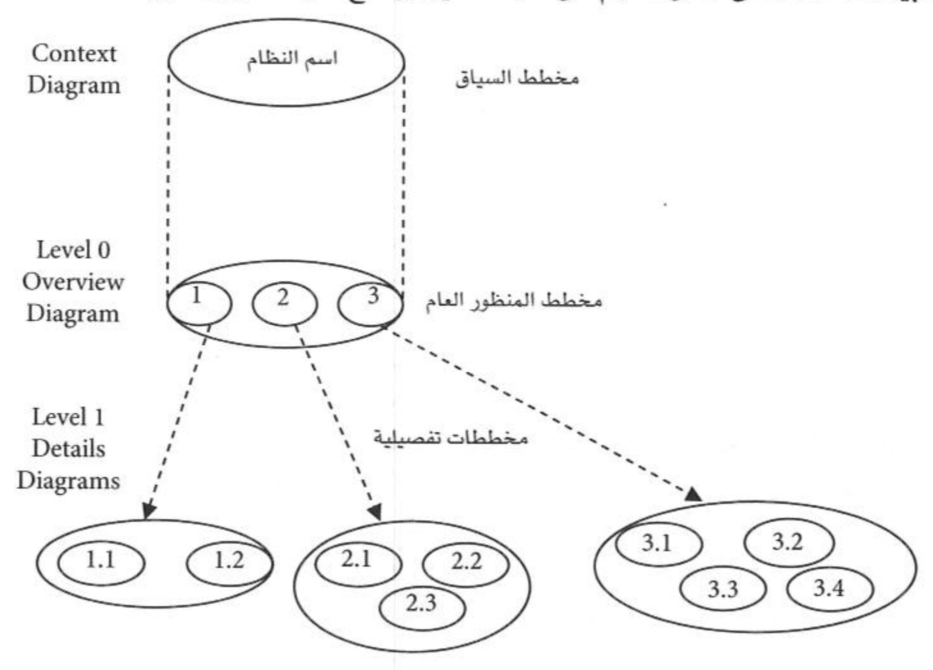
تحتوى مخططات تدفق البيانات على مجموعة من العناصر الأساسية التى تستخدم لوصف عمليات النظام فى المستويات المختلفة وهى (السماك، ٢٠٠٣):

١- الكائنات الخارجية External Entities:

تمثل مجموعة الأشــخاص والمؤسسات والأقسام والأنظمة الأخرى التى تتفاعل مع النظام من الخارج وتمده بالبيانات وتستقبل منه مخرجات.

٢- تدفق البيانات Data Flow:

تمثل عملية نقل أو سريان البيانات من عنصر إلى عنصر آخر داخل مخطط تدفق البيانات DFD فى شكل أسهم فى اتجاه معين يوضح اتجاه سريان البيانات.



شكل (٢-١) التفكيك الوظيفى فى مخططات DFDs

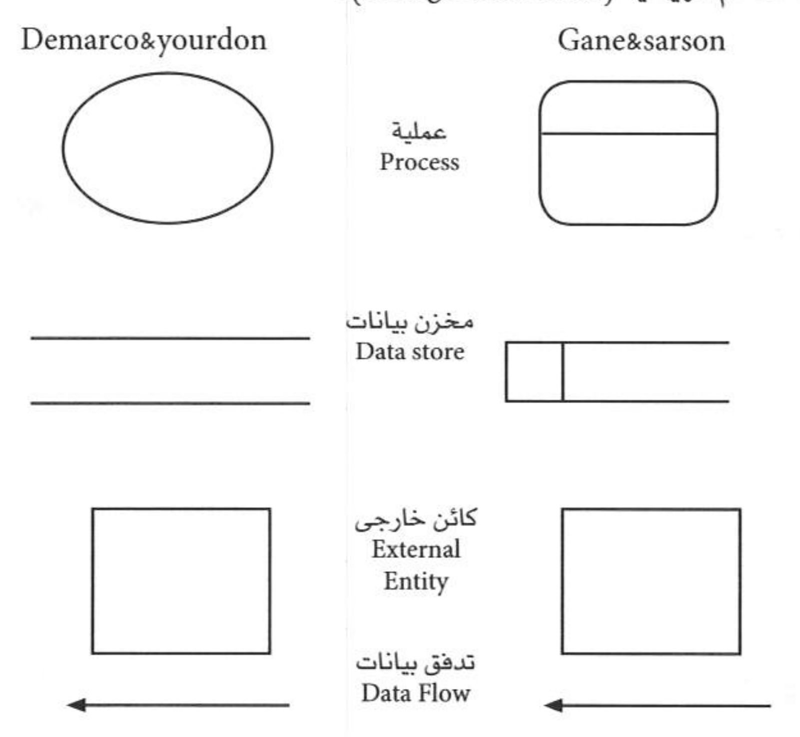
٣- العمليات Processes:

تمثل مجموعة الأنشــطة التى تقــوم بمعالجة البيانات فى النظــام للحصول على المخرجات. هناك نوعان من العمليات: عمليات أولية أو أساسية، وعمليات فرعية أو تفصيلية للعمليات الأساسية.

٤- مخازن البيانات Data Stores:

هو مستودع يحتوى على بيانات كائنات النظام وكذلك تدفقات البيانات.

يوضح شكل (٢-٢) الرموز المستخدمة لكل عنصر من عناصر مخططات تدفق البيانات فى كل من أسلوب "Yourdon & DeMarco" وأسلوب "Gane & Sarson" لنمذجة النظم الهيكلية (George et al., 2002).

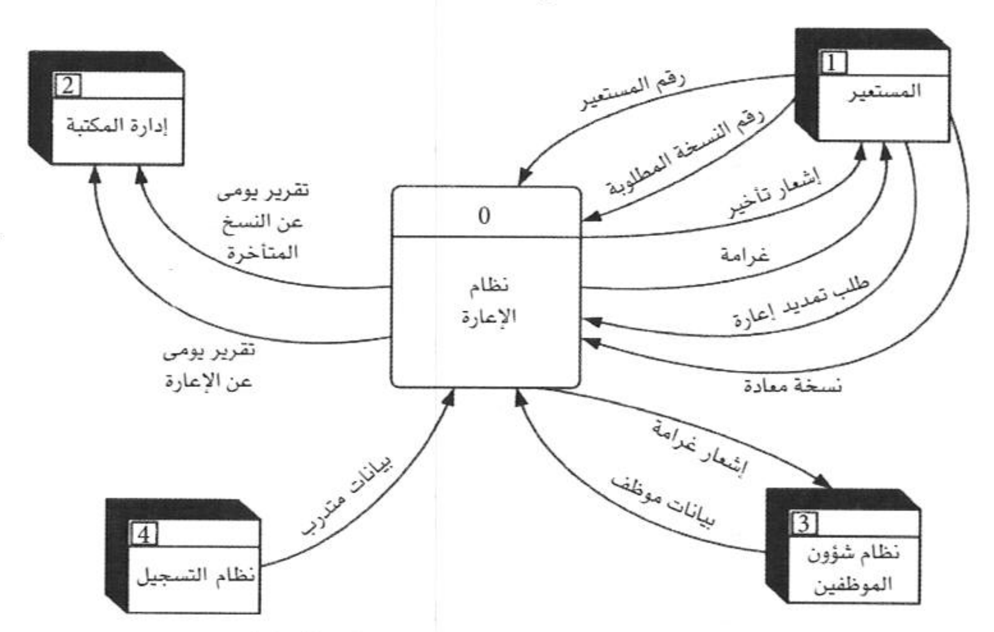


شكل (٢-٢) الرموز المستخدمة فى مخططات تدفق البيانات

٢-١-١-١ مخطط السياق Context Diagram:

الغرض من مخطط السياق هو تحديد موقع نظام الأعمال داخل البيئة المحيطة به. يقوم المخطط أيضاً بتعريف حدود النظام وتعريف تفاعل النظام مع الكائنات الخارجية له من خلال مجموعة التدفقات القادمة منها والمرسلة إليها، وهى تمثل مدخلات ومخرجات النظام. يتكون مخطط السياق للنظام من عملية واحدة تمثل

منظراً عاماً للنظام من الخارج وتفاعله مع بيئة العمل المحيطة به من خلال التدفقات بينه وبين الكائنات الخارجية له. يوضح شكل (٢-٣) مخطط السياق لنظام الإعارة بمكتبة معهد الإدارة العامة. يمثل كل من المستعير ونظام التسجيل ونظام شؤون الموظفين وإدارة المكتبة، الكائنات الخارجية للنظام أى أنها تمثل بيئة النظام. وتمثل التدفقات مثل «رقم المستعير»، و«طلب تمديد إعارة»، و«بيانات موظف»، التفاعلات المختلفة بين الكائنات الخارجية والنظام. سيتم عرض جميع العمليات الموجودة داخل النظام من خلال مخطط تدفق البيانات DFD.



شكل (٢-٣) مخطط السياق لنظام الإعارة

٢-١-١-٢ مخطط المنظور العام Overview DFD:

يتم توصيف ما يحدث داخل مخطط السياق عن طريق مخطط المنظور العام، وهو يعتبر «المستوى صفر» لمخططات تدفق البيانات DFD Level-0. يركز هذا المخطط على وصف العمليات الرئيسية للنظام وتدفق البيانات خلالها، وهو يحتوى على ما يلى:

- الكائنات الخارجية External Entities.
- العمليات الأساسية Main Processes.
- مخازن البيانات Data Stores.
- تدفقات البيانات Data Flows.

يوضح شكل (٢-٤) مخطط المنظور العام لنظام الإعارة بمكتبة معهد الإدارة العامة. يحتوى المخطط على أربع عمليات رئيسية هى:

١- «أنشئ ملف مستعير» وهى العملية المسئولة عن إنشاء ملف المستعيرين بالمكتبة وذلك بناء على البيانات المأخوذة من نظام القبول والتسجيل (بيانات المتدربين والطلاب) ونظام شئون الموظفين (بيانات الأساتذة والموظفين).

٢- «نفذ إعارة» وهى العملية المسئولة عن استقبال طلب الاستعارة من المستعير والتحقق من إمكانية تنفيذه، ثم تسجيل بيانات الإعارة فى ملف الإعارة، وتحديث بيانات الكتاب المعار، وعدد الكتب المعارة للمستعير.

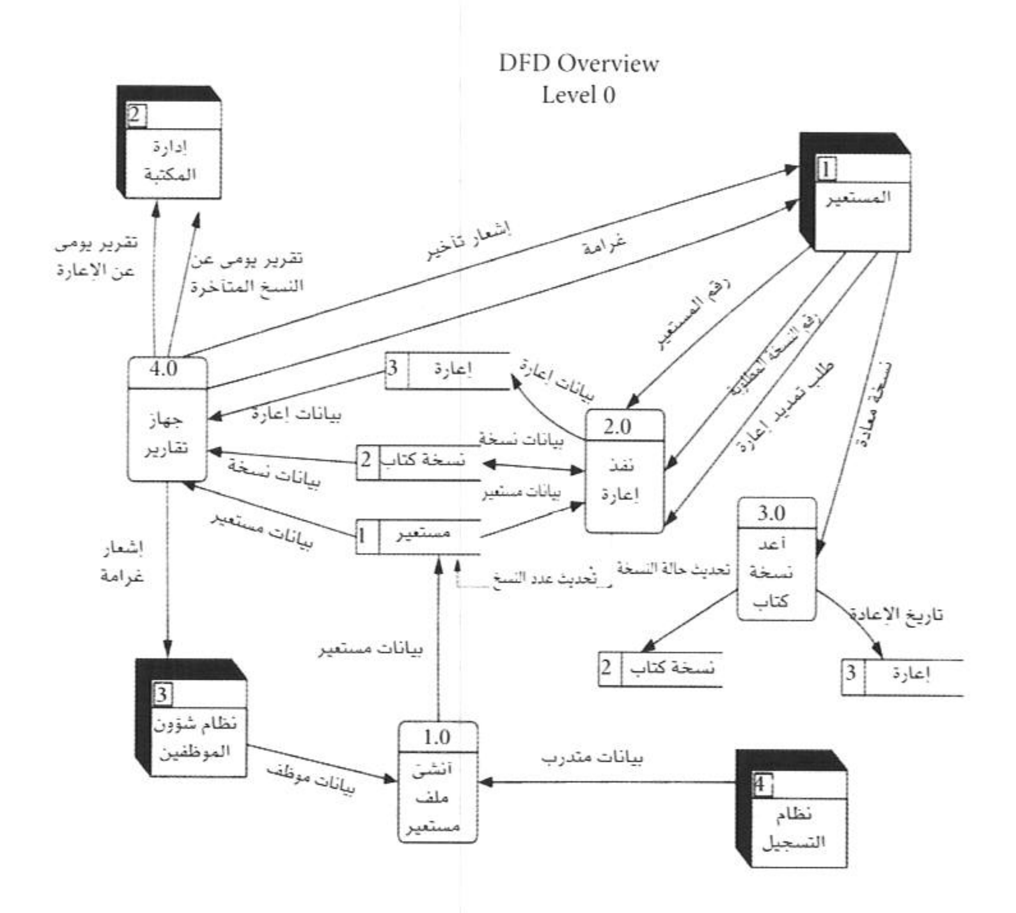
٣- «أعد نسخة كتاب» وهى العملية المسئولة عن تسجيل بيانات إعادة نسخة الكتاب المعارة إلى المكتبة، وتحديث بيانات الإعارة، وحالة الكتاب، وعدد الكتب المعارة للمستعير.

٤- «جهز تقارير» وهى العملية المسئولة عن إعداد جميع التقارير المطلوبة لإدارة النظام من حيث تصميم الشكل المطلوب للتقرير وتكوين جملة الاستعلام المناسبة لتنفيذ التقرير. بالإضافة إلى إصدار إشعارات التأخير والغرامات للمستعيرين المتأخرين عن موعد إعادة الكتب المعارة لديهم أو فقدانها.

ويحتوى المخطط أيضاً على ثلاثة مخازن للبيانات هى «مستعير»، و«نسخة كتاب»، و«إعارة». نلاحظ فى الشكل أن التدفقات الخارجية للنظام هى التدفقات الخارجية الموجودة نفسها بمخطط السياق، ولكن هناك تفاصيل أدق لتدفقات أخرى داخلية بين العمليات ومخازن البيانات.

يجب أن يراعى عند تصميم مخطط المنظور العام لتدفق البيانات أن تكون الاعتمادية بين العمليات الرئيسية التى يتضمنها أقل ما يمكن، بحيث لا تعتمد عملية ما على عملية أخرى عند تنفيذها. فمن المفترض أن تكون الوسيلة الطبيعية لبدء تشغيل

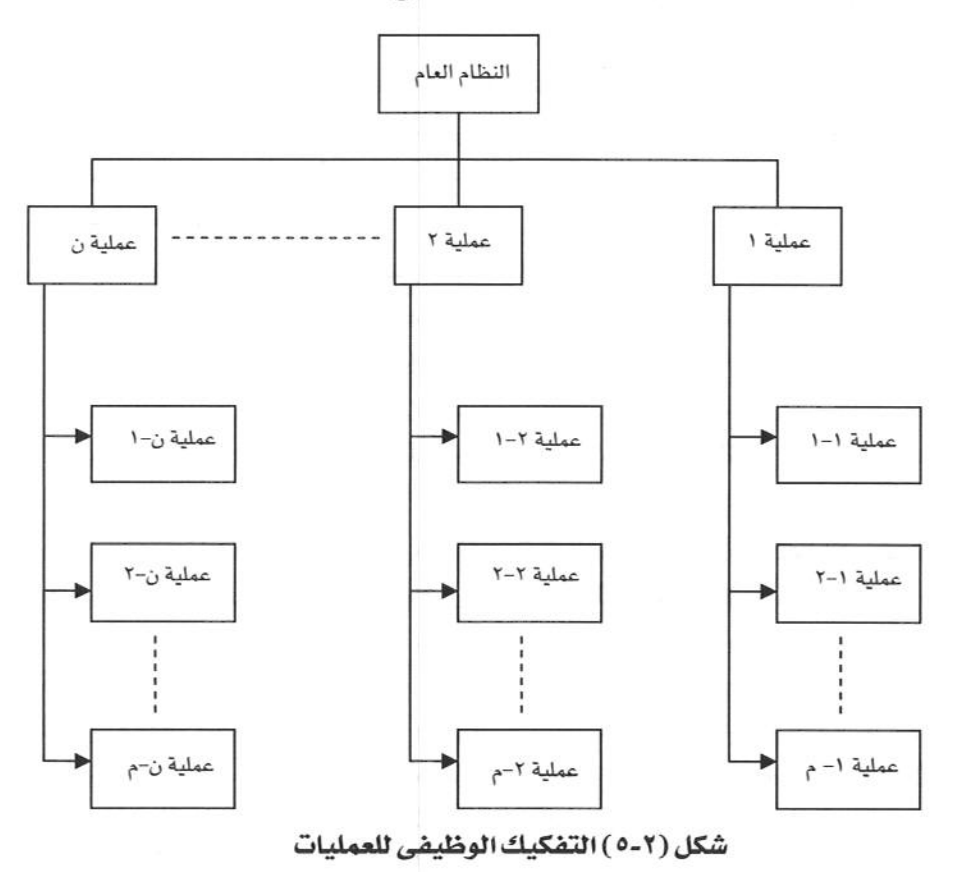
أى عملية رئيســية من خلال الكائنــات الخارجية التى تتفاعل مع النظام أو من خلال ميقات زمنى محدد. وللتغلب على مشكلة الاعتمادية التى يمكن أن تنشأ بين العمليات الرئيســية بالنظام، يتم استخدام مخازن البيانات كوســيط طبيعى لتبادل المعلومات. فيمكــن لأى عملية أن تســتقبل مدخلاتها وتعالجها ثم تقوم بتســجيل مخرجاتها فى مخــازن البيانات، ومن ثم يمكــن لأى عملية أخرى بالنظام اســتخدام تلك المعلومات المخزنة باعتبارها مدخلات طبيعية لها.



شكل (٢-٤) مخطط المنظور العام لنظام الإعارة

٢-١-١-٣ المخططات التفصيلية Detailed DFDs:

يتم توصيف مجموعة الأنشطة التى تتضمنها العمليات الأساسية للنظام من خلال مجموعة من المخططات التفصيلية لمكونات مخطط المنظور العام، وذلك فى عدة مستويات داخلية إلى أسفل. يتم تفكيك العملية الأساسية فى مخطط المنظور العام إلى مجموعة من العمليات الفرعية المترابطة فى شكل مخطط تدفق بيانات منفصل لهذه العملية فى مستوى تالٍ. تبدأ المخططات التفصيلية بالمستوى واحد DFD Level-1. وإذا كانت إحدى هذه العمليات الفرعية تحتوى على تفاصيل أخرى، فإنه يتم تفكيكها فى شكل مخطط تدفق بيانات منفصل فى المستوى الثانى DFD level-2، وهكذا. يعرف هذا الأسلوب بأسلوب «التفكيك الوظيفى Functional Decomposition"، كما هو موضح بالشكل (٢-٥).



شكل (٢-٥) التفكيك الوظيفى للعمليات

ويوضح شكل (٢-٦) مخطط تدفق البيانات، المستوى واحد DFD Level-1 للعملية رقـم ٢ «نفذ إعارة» فى مخطط المنظور العام لنظام الإعارة الموضح بالشـكل (٢-٤). نلاحظ من الشكل أن العمليات الفرعية تتبع نظام ترقيم يبدأ برقم العملية الأساسية ثم الرقم الفرعى مثل ٢-١، ٢-٢، ٢-٣ وهكذا. تحتوى العملية الأساسية «نفذ إعارة» على أربع عمليات تفصيلية هى:

٢-١ «حدد المسـتعير» وهى المسئولة عن استخراج بيانات المستعير من ملف المستعير بناء على البيان المدخل لها وهو رقم المستعير.

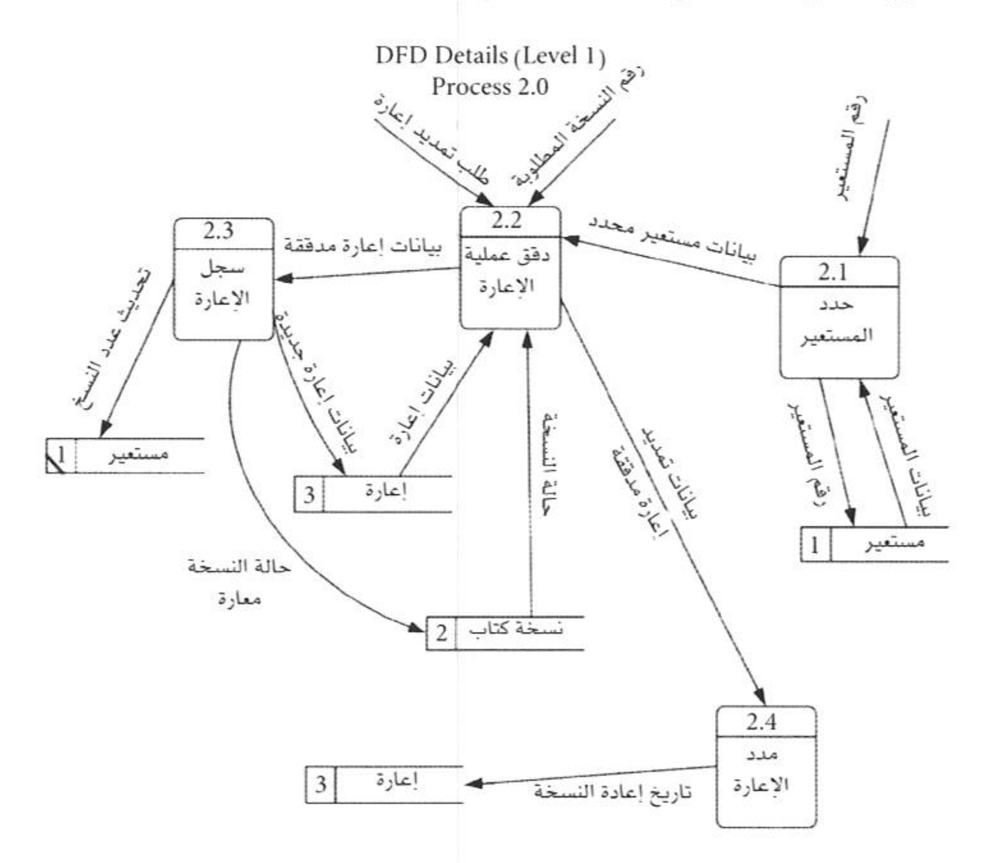
٢-٢ «دقق عملية الإعارة» وهى المسـئولة عن تدقيق بيانـات الإعارة المطلوبة، فتبدأ بتحديد نوع العملية المطلوبة «إعارة جديدة/تجديد إعارة»، وبعد ذلك يتم التحقق من حالة نسـخة الكتاب المطلوبة للإعارة «متاحة/معارة» وذلك باستخدام بيان رقم نسخة الكتاب. وإذا كانت نسخة الكتاب متاحة للإعارة فإنه يتم التأكد من أحقية المستعير لإتمام عملية الإعارة وذلك من خلال معرفة عدد الكتب المعارة للمسـتعير ومقارنتها بالعدد المسـموح به، والتأكد من عدم وجود محاذير على المسـتعير نفسه كأن يكون لديه كتب متأخرة أو مفقودة (أى أنه موجود بالقائمة السوداء).

٢-٣ «سجل الإعارة» وهى المسئولة عن تنفيذ عملية الإعارة من عدمها بناء على البيانات المقدمة لها من العملية السـابقة، فيتم تسـجيل بيانات الإعارة فى ملف الإعارة وتحديث عدد النسخ المعارة للمستعير وتحديث حالة نسخة الكتاب المعار.

٢-٤ «مدد الإعارة» وهى المسئولة عن تحديث بيانات الإعارة (تاريخ الاسترجاع لنسخة الكتاب المعار) فى حالة طلب المستعير تمديد فترة الإعارة لنسخة الكتاب المعار لديه.

ونلاحظ أيضاً أن التدفقات الخارجية للمخطط التفصيلى هى نفسـها التى تصف حدود العملية الأساسـية «نفذ إعارة» فى مخطط المنظور العام وهى نفسـها الموجودة بمخطط السـياق، وذلك لضمان تحقيق الاتسـاق التام بين مخططات وصف النظام فى المسـتويات المختلفـة. بالإضافة إلى وجود تدفقات أخـرى داخلية بين العمليات التفصيليـة وبين مخازن البيانـات. ومن الممكن أن تكون هنـاك عملية تفصيلية فى هذا المسـتوى Level 1 تحتاج إلى تفاصيل أكثر. وهنا يجب أن يتم وصفها من خلال مخطط منفصل فى المسـتوى الثانى Level 2 ويتضمن عمليـات تبدأ بترقيم العملية السابقة فى المستويين السابقين مثل ٢-١-١، ٢-١-٢، وهكذا.

ويجـب أن يراعى أيضاً فى أثنـاء تصميم المخططات التفصيليـة لتدفق البيانات أن يكـون الارتبـاط بين العمليات التفصيليـة عالياً جداً. بمعنـى أن تكون العمليات التفصيليـة فى حالة تفاعل وارتبـاط طبيعى لتحقيق الهدف من العملية الرئيسـية. ولا يتحقـق ذلك إلا من خـلال التفاعل الطبيعى بين العمليـات التفصيلية، كأن تقوم عملية باسـتقبال بيانات مدخلة وتمررها إلى عملية أخـرى تفصيلية لتدقيقها، ومن ثم تقوم عمليـة أخرى تفصيلية بمعالجتها، ثم تمرر مخرجاتها إلى عملية أخرى تقوم بتخزين المعلومات فى مخازن البيانات، كما هو مبين بالشكل (٢-٦) لتوضيح العمليات التفصيلية لعملية «نفذ إعارة».



شكل (٢-٦) مخطط تدفق البيانات التفصيلى لعملية «نفذ إعارة»

٢-١-٢ وصف منطق العمليات Process Logic Description:

تتطلـب عملية توصيف المتطلبات الوظيفية للنظام أن يتم توصيف منطق العمليات الفرعية للنظام فى شـكل نمـاذج نصية، ويتم إرفاقها مع مخططـات تدفق البيانات للنظام. الهدف من هذه النماذج هو وصف إجراءات العمل لكل عملية بشـكل منطقى لا يعتمد على الأدوات أو التقنية التى سـتقوم بتنفيذها. هناك عدة طرق تسـتخدم لوصف منطق العمليات منها:

- الإنجليزية الهيكلية Structured English أو العربية الهيكلية.
- مخططات العمل Action Diagrams.
- جداول القرارات Decision Tables.
- شجرة القرارات Decision Tree.

ومن الضرورى أن تتسـم عملية توصيف منطق العمليات بالأسـلوب المنطقى فقط ولا تسـتخدم أسـلوب لغة برمجة معينة ولكـن تتضمن المفاهيم الأساسـية للبرمجة الهيكلية. وتعتبر هذه النماذج النصية أداة تفاهم مشتركة بين أفراد فريق التطوير من المحللين والمصممين والمبرمجين.

٢-١-٢-١ الإنجليزية الهيكلية Structured English:

يفضل أغلب المحللين اسـتخدام أسـلوب الإنجليزية (أو العربية) الهيكلية لوصف منطق العمليات وذلك لتميزها بالجمع بين أسـلوب اللغة الطبيعية والأسلوب الهيكلى الذى يلائم المنطق فى أثناء صياغة الجمل التى تصف خطوات تنفيذ كل عملية. يتسم هذا الأسلوب بعدة مميزات منها:

- سهولة وصف الإجراءات وذلك باستخدام لغة طبيعية.
- عدم التقيد باستخدام أسلوب لغة برمجة معينة.
- المرونـة فى اسـتخدام الهيـاكل البرمجية الأساسـية مثل التدفقـات والتفريعات والتكرارات.
- تسهيل عملية التفاهم والاتصال بين فرق العمل بالمشروع (تحليل، تصميم، برمجة).
- سهولة صيانة تفاصيل العمليات.

يفضل بعض أفراد التحليل استخدام أسلوب مخططات العمل أثناء وصف منطق العمليات وذلك لقربها من الأسلوب البرمجى ولكن دون التقيد بلغة برمجة معينة. يفضل استخدام هذا الأسلوب فى حالة اشتراك أفراد فريق العمل فى أكثر من نشاط (تحليل، تصميم، برمجة) فى أثناء عملية التطوير. ولكن يعيب هذا الأسلوب صعوبة صيانة منطق العمليات فيما بعد خاصة من جانب أفراد آخرين التحقوا بالعمل فى المشروع. ولذلك فإنه من الأفضل استخدام أسلوب الإنجليزية الهيكلية مع مراعاة تغليب الطابع الهيكلى على الطابع الإنشائى للجمل المستخدمة لوصف منطق العملية. يؤدى هذا الأسلوب إلى الجمع بين مميزات الأسلوبين من حيث سهولة الفهم واستخدام التركيبات المنطقية المشابهة للتركيبات البرمجية. ومع ذلك فهناك بعض أفراد فريق التحليل يفضلون استخدام أسلوب مخطط التدفق Flow Chart لوصف منطق العمليات بشكل يجمع بين الأسلوب التخطيطى والأسلوب البرمجى. يوضح شكل (٢-٧) مثالاً على استخدام أسلوب العربية الهيكلية لوصف منطق العملية الفرعية «سجل الإعارة» فى نظام الإعارة.

إذا كان ناتج العملية ٢-٢ = "إعارة"
  سجل بيانات إعارة جديدة فى سجل الإعارة
  حدث حالة نسخة الكتاب لتصبح " معار"
  حدث عدد النسخ المعارة للمستعير بإضافة واحد
أما إذا كان ناتج العملية ٢-٢ = "حجز"
  فيتم تسجيل بيانات نسخة الكتاب والمستعير فى ملف الحجز
وإلا فيتم "رفض" تنفيذ عملية الإعارة

شكل (٢-٧) مثال على استخدام أسلوب العربية الهيكلية

وفى حالة احتواء العملية على عدد كبير من الشروط والأفعال مما يؤدى إلى كثرة التفريعات، ومن ثم صعوبة فى الوصف أو الفهم، فإنه يفضل استخدام أسلوب جداول القرارات أو شجرة القرارات. يتميز هذان الأسلوبان بدقة وسهولة التعامل مع التفريعات الناتجة عن تعدد الشروط وتداخلها وذلك ضمن قواعد العمل التى تحكم إجراءات عملية معينة تتصف بالتعقيد أو صعوبة التركيب. وفيما يلى عرض بسيط

لاســـتخدام الأســـلوبين فى وصف منطق عملية «دقق عملية الإعارة» فى نظام الإعارة بمكتبة معهد الإدارة العامة.

٢-١-٢-٢ جداول القرارات Decision Tables:

يتم أولاً تحديد مجموعة الشروط والأفعال الخاصة بعملية التدقيق وهى كما يلى

الشروط:

١- هل المســـتعير موجـــود بالقائمة الســـوداء نتيجة فقدان كتب معـــارة لديه أو كثرة التأخير فى استرجاع الكتب المعارة؟

٢- هل عدد الكتب المعارة حالياً للمستعير أقل من الحد الأقصى المسموح به؟

٣- هل نسخة الكتاب المطلوب إعارتها متاحة للإعارة أو لا؟

الأفعال:

١- إعارة (تنفيذ عملية الإعارة للمستعير).

٢- حجز (حجز نســـخة الكتاب للمستعير فى حالة عدم توافر أى نسخة الآن للإعارة أو فى حالة وصول عدد الكتب المعارة للمستعير للحد الأقصى).

٣- رفض (رفض تنفيذ عملية الإعارة للمستعير نظراً لوجوده ضمن القائمة السوداء).

نقوم الآن بتمثيل تلك المعلومات من خلال جدول القرارات الذى يأخذ شكل الجداول المنطقية (شكل (٢-٨)) كما يلى:

شرط ١	شرط ٢	شرط ٣	الأفعال
لا	لا	لا	رفض
نعم	لا	لا	رفض
لا	نعم	لا	حجز
نعم	نعم	لا	رفض
لا	لا	نعم	حجز
نعم	لا	نعم	رفض
لا	نعم	نعم	إعارة
نعم	نعم	نعم	رفض

شكل (٢-٨) الشكل الأولى لجدول القرارات لعملية «دقق عملية الإعارة»

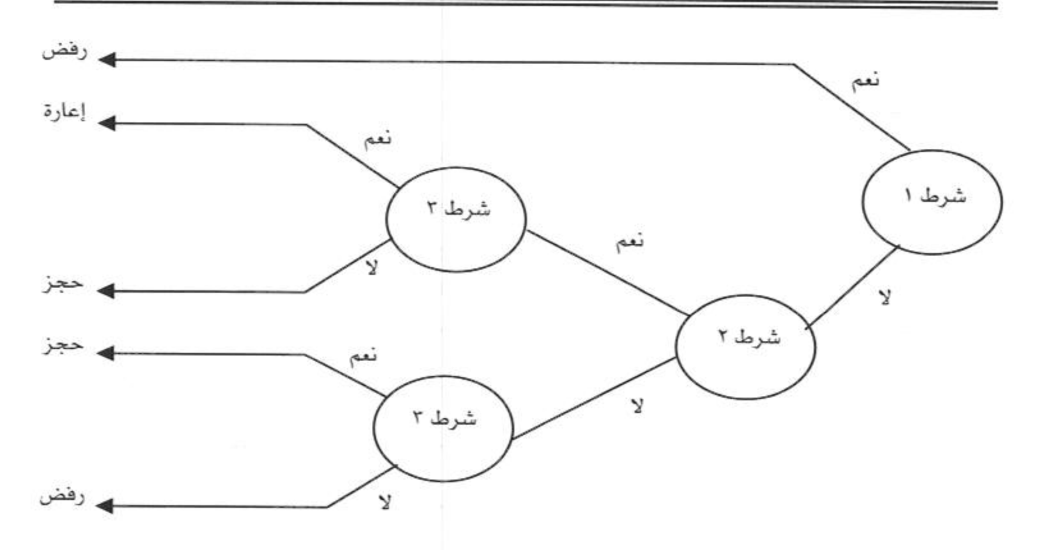
نلاحظ من الجدول أنه يتم تنفيذ عملية الإعارة فى حالة واحدة فقط، والحجز فى حالتين، والرفض فى باقى الحالات. وبدراسة حالات الرفض نجد أن عددها خمس حالات، منها أربع حالات تتضمن الشرط ١ عندما تكون قيمته تساوى «نعم». ولذلك فسوف نقوم بتمثيل الحالات الأربع بحالة واحدة تحتوى على قيمة الشرط الأول فقط بغض النظر عن قيم الحالات الأخرى. يؤدى ذلك إلى إنتاج شكل مبسط لجدول القرارات كما هو مبين بالشكل (٢-٩).

شرط ١	شرط ٢	شرط ٣	الأفعال
لا	لا	لا	رفض
نعم	-	-	رفض
لا	نعم	لا	حجز
لا	لا	نعم	حجز
لا	نعم	نعم	إعارة

شكل (٢-٩) الشكل المبسط لجدول القرارات لعملية «دقق عملية الإعارة»

٢-١-٢-٣ شجرة القرارات Decision Tree:

يمكننا تمثيل المعلومات السابقة نفسها بأسلوب آخر يأخذ شكل شجرة المعلومات وهو ما يسمى بشجرة القرارات. يبدأ جذر الشجرة بالشرط الأول وتكون فروعه هى القيم المسموح بها للشرط، وتكون محصورة فى قيمتين فقط بالنسبة للشروط المنطقية هما (نعم/لا). وينطلق الشرط الثانى من نهاية فرعى الشرط الأول مما يؤدى إلى أربعة فروع. وينطلق الشرط الثالث من نهاية الفروع الأربعة السابقة، مما يؤدى إلى ثمانية أفرع، وهكذا لباقى الشروط حتى نصل إلى القرارات التى تمثل العملية. يمكننا أيضاً تبسيط شجرة القرارات بدمج الفروع التى تؤدى إلى القرار نفسه وتعتمد على شرط معين أو أكثر لاتخاذ القرار بغض النظر عن قيمة باقى الشروط. يوضح شكل (٢-١٠) الشكل المبسط لشجرة القرارات المقابل لجدول القرارات المبسط (شكل (٢-٩).



شكل (٢-١٠) الشكل المبسط لشجرة القرارات لعملية «دقق عملية الإعارة»

٢-١-٣ مخطط العلاقة بين الكائنات (Entity Relationship Diagram (ERD:

فــى حين كانت تركز مخططات تدفق البيانــات DFDs على وصف عمليات نظام الأعمــال، فإن مخططات العلاقة بــين الكائنات ERDs تركز على وصف كائنات نظام الأعمال وتمثيل العلاقات بينها، فيما يعرف بنمذجة بيانات النظام Data Modeling. يســاهم أســلوب نمذجة بيانات النظام فى تنظيم وتوثيق جميع بيانات نظام الأعمال، ويعتبــر هو المدخل الطبيعى لتصميم قواعد بيانات النظام. تتم نمذجة بيانات النظام بشــكل مرئى من خلال تمثيــل جميع كائنات النظام Entities بشــكل منطقى وتمثيل العلاقات المنطقية بينها Relationships بغض النظر عن التقنية المستخدمة لتنفيذها. يتضمن نموذج ERD ما يلى (السماك، ٢٠٠٢):

٢-١-٣-١ كائنات النظام System Entities:

يمثل الكائن Entity أى شــىء لــه دور فعال فى نظام الأعمال ويتطلب تســجيل بياناته فى مخازن بيانات النظام. فيما يلى أمثلة لبعض الكائنات الشائعة الاستخدام فى نظم الأعمال.

شخص (عميل، طالب، مدرس، موظف،.......).

شىء (كتاب، سلعة، جهاز، سيارة، عقار،......).

مكان (منطقة، مخزن، فرع، مبنى،........).

حدث (بيع، شراء، تسجيل، حجز، طلب،......).

مفهوم (حساب، مخزون، منهج دراسى، اعتماد مالى، ....).

خصائص الكائنات Entities Attributes:

تصف خصائص الكائن مجموعة الصفات أو السمات التى تميزه عن باقى الكائنات مثل الاسم، العنوان، الوظيفة، وغيرها. أحياناً يحتوى الكائن على خصائص مركبة، أى أنها تحتوى على خصائص أخرى فرعية مثل:

الاسم (الأول، المتوسط، الأخير).

العنوان (الدولة، المدينة، الرمز البريدى، صندوق البريد).

يحتوى كل كائن على خاصية أو أكثر تميز حالات الكائن Instances، أى أنها تأخذ قيمة وحيدة Unique Value لا يمكن أن تتكرر فى حالات أخرى للكائن. تستخدم هذه الخاصية لتحديد هوية الكائن أو ما يطلق عليه «مفتاح أساسى Primary Key". إذا كان هناك أكثر من خاصية يمكن أن تستخدم مفتاحاً أساسياً، فيتم اختيار أحدها ويكون الآخر مفتاحاً بديلاً Alternate Key يمكن استخدامه عند الضرورة فيما بعد أثناء صيانة أو توسعة نظام الأعمال. وهناك خصائص أخرى يتكرر حدوثها فى حالات عديدة للكائن وتكون غالباً مماثلة لقيم مفتاح أساسى فى كائن آخر، ويطلق عليها «مفاتيح خارجية Foreign Keys". يستخدم هذان النوعان من المفاتيح لتوصيف العلاقات المختلفة بين كائنات النظام. يوضح شكل (٢-١١) مثالاً لخصائص كائن «طالب».

Student

Student Number (Primary Key)
National Number (Atternatekey)
Name (Composite Attribute)
- Last Name.
- First Name
- Middle Name

Address (Comeosite Attribute)
- Street
- City
- State
- Postal code

Telephone
Birth Date
Gender (Subsetting Criteria1)
Major (Subsetting Criteria 2)
Grade Point Average

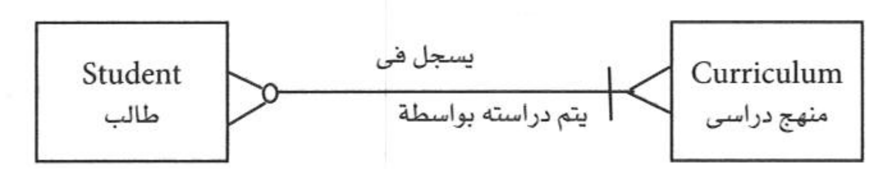
شكل (٢-١١) خصائص كائن «طالب»

٢-١-٣-٢ العلاقة بين الكائنات Entities Relationships:

العلاقة هى رابطة عمل طبيعية موجودة بين واحد وأكثر من الكائنات داخل النظام. من الممكن أن تمثل العلاقة حدثاً Event معيناً يربط بين الكائنات أو تمثل علاقة قرابة منطقية موجودة أصلاً بين الكائنات. فمثلاً تكون العلاقة المنطقية التى تربط بين المنهج الدراسى والطالب نتيجة حصول حدث معين وهو إجراء عملية التحاق الطالب بهذا المنهج الدراسى. وتكون العلاقة بين المدرس والمادة من خلال عملية التدريس أو التأهيل للتدريس. وتكون العلاقة بين المسافر ورحلة طيران معينة من خلال عملية الحجز، وهكذا. وكمثال للعلاقة بين الكائنات نتيجة القرابة الطبيعية بينها، نجد العلاقة الطبيعية بين العقار ومالك العقار، إذ إن لكل عقار مالكاً معيناً، والعلاقة بين الأب والتابعين له فى الملف الصحى أو الاجتماعى، وهكذا.

٢-١-٣-٢-١ تعددية العلاقة Cardinality:

مــن الطبيعى أن يكون لأى علاقة اتجاهان (اتجــاه لكل طرف من طرفى العلاقة) ويجــب تحديد نوع العلاقة لكل اتجاه. تبين تعددية العلاقة العدد الأكبر Maximum والعدد الأصغر Minimum لتكرار حدوث حالات أحد الكائنات المتعلقة بحدوث حالة كائــن آخــر مرتبط به. يجب تحديد التعددية فى كلا الاتجاهين للعلاقة نظراً لأن كل العلاقات ثنائية الاتجاه. فمثلاً العلاقة بين الطالب والمنهج الدراســى تعتبر من النوع «واحد أو أكثر إلى صفر أو أكثر Many-To-Many"، فمن جهة كائن الطالب نجد أن الطالب الواحد يمكن أن يلتحق بمنهج دراسى أو أكثر. ومن جهة كائن المنهج الدراسى نجد أن المنهج الواحد يمكن أن يتم دراســته من قبل طالب أو أكثر أو لا يتم دراســته حتى الآن، كما هو مبين بالشكل (٢-١٢).



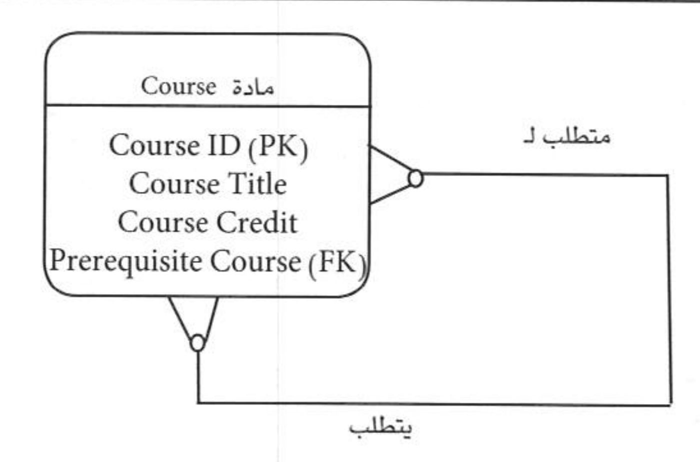
شكل (٢-١٢) تمثيل العلاقة بين الكائنات

٢-١-٣-٢-٢ درجة العلاقة Degree of Relationship:

تمثل درجة العلاقة عدد الكائنات التى تشارك فى العلاقة وهى ثلاثة أنواع:

١- علاقة ذاتية Recursive Relationship **أو أحادية** Unary:

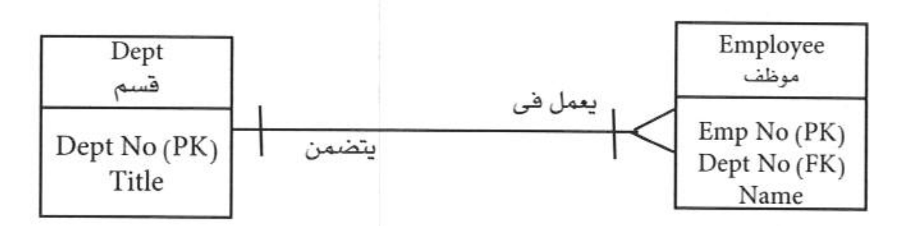
هى علاقــة بين حالات Instances من الكائن نفســه، وفيها يحتــوى الكائن على خاصيــة (حقل) ترتبط بخاصية أخرى موجودة كما فى المثال التالى (شــكل (٢-١٣)). فنجــد أن المادة Course يمكن أن تتطلب إتمام دراســة مــادة أخرى أو أكثر كمتطلب أساسى لتدريسها. يتم وصف تلك العلاقة بين رقم المادة Course ID (المفتاح الأساسى للكائن) وبين رقم المادة المتطلبة Prerequisite Course (المفتاح الخارجى داخل الكائن نفســه). وبالمثل فإنه مــن الممكن أن تكون المادة متطلباً أساســياً لمادة أخرى أو أكثر. ولذلــك فإن العلاقة تأخذ الشــكل متعدد - إلى - متعدد ومن النــوع «اختيارى» فى الاتجاهين.



شكل (٢-١٣) العلاقة الذاتية

٢- علاقة ثنائية Binary Relationship:

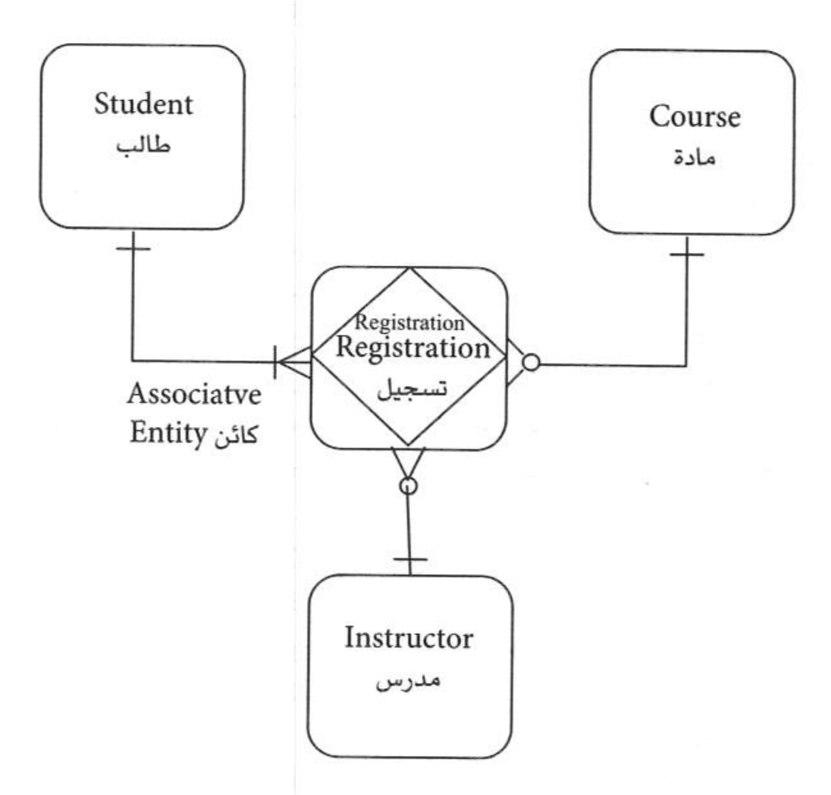
هـــى علاقة بين حـــالات كائنين، يتمثل الارتباط فيها من خـــلال خاصية (أو أكثر) مشـــتركة فى الكائنين وتكون تلـــك الخاصية هى المفتاح الأساســـى فى أحد الكائنين وتكـــون المفتاح الخارجى فى الكائن الآخر المرتبط به. يوضح الشـــكل (٢-١٤) إحدى العلاقات الثنائية التى تمثل العلاقة بين الموظف Employee والقسم Dept الذى يعمل به. فنجد أن كل قسم يحتوى على مجموعة من الموظفين العاملين به، وأن كل موظف يجب أن يكون ملتحقاً بقســـم معـــين. أى أن العلاقة من النـــوع واحد - إلى- متعدد من ناحية القســـم باتجاه الموظف. ولذلك فقد تم إدارج رقم القســـم ضمن خصائص الموظف باعتباره مفتاحاً خارجياً يرتبط بكائن القســـم Dept ليصف عملية الربط بين الكائنين.



شكل (٢-١٤) العلاقة الثنائية

٣- علاقة متعددة N-ary Relationship:

تعتبر العلاقات الثنائية من أوضح أنواع العلاقات بين الكائنات وأسهلها عند التطبيق من خلال أحد نظم إدارة قواعد البيانات المتاحة. ولكن كما ذكرنا من قبل فإن تمثيل العلاقات بين الكائنات يكون ترجمة حقيقية لطبيعة الارتباط الفعلى بين تلك الكائنات. وأحياناً يكون الارتباط ممثلاً من خلال علاقة مركبة بين أكثر من كائنين، كما فى المثال الموضح بالشكل (٢-١٥) الذى يوضح علاقة مركبة بين ثلاثة كائنات هى «الطالب» و«المدرس» و«المادة» وهى تحدث فعلياً فى أثناء عملية التسجيل للطالب فى مادة معينة قبل بداية الفصل الدراسى فى نظام التسجيل بالجامعات بأسلوب الساعات المعتمدة. ونلاحظ أن تلك العلاقة تحتوى على خصائص معينة تصف عملية التسجيل نفسها مثل «تاريخ التسجيل» و«المعدل الفصلى». ويتم تمثيل تلك العلاقة المركبة من خلال كائن آخر يمثل عملية الارتباط بينهم يسمى «كائن الربط Associative Entity" وهو يستخدم للربط بين أكثر من كائن، وهو يحتوى على مفتاح أساسى مركب موروث من الكائنات الأخرى المرتبطة من خلاله. يشير كل جزء من المفتاح الأساسى المركب لكائن الربط إلى مفتاح أساسى لكائن من الكائنات المرتبطة، وهو يمثل أحد المفاتيح الخارجية لكائن الربط. ويتضمن أيضاً كائن الربط الخصائص الإضافية التى تخص العلاقة الناتجة عن الحدث وهو «التسجيل» مثل «تاريخ التسجيل» و «المعدل الفصلى». والجدير بالذكر هنا أنه لا ينبغى أن نستبدل تلك العلاقة الثلاثية الصريحة بمجموعة من العلاقات الثنائية البسيطة كحل للمشكلة. كأن يتم تمثيل العلاقة الثلاثية من خلال ثلاث علاقات ثنائية بين كل من «الطالب-المادة» و«الطالب-المدرس» و«المدرس-المادة». إذ إن تلك العلاقات هى بمنزلة علاقات افتراضية غير واقعية مطلقاً. فمثلاً لا توجد علاقة طبيعية بين الطالب والمادة إلا من خلال عملية التسجيل، وكذلك لا توجد علاقة طبيعية بين الطالب والمدرس إلا من خلال عملية التسجيل أو التدريس. وأيضاً لا توجد علاقة طبيعية بين المدرس والمادة إلا من خلال عملية التسجيل أو التدريس. ولذلك فإن عملية التسجيل أو التدريس هى العلاقة الطبيعية التى تربط بين الكائنات الثلاثة فى الوقت نفسه. وبالمثل نجد أن هذا المفهوم ينطبق على العديد من التطبيقات العملية مثل قيام الموظف بعملية حجز موعد لمريض فى عيادة معينة عند طبيب محدد، أو قيام موظف معين بتسجيل عملية حجز لمسافر فى رحلة طيران معينة، أو تكليف موظف بتنفيذ مهمة محددة فى مشروع معين يتطلب مهارة محددة، وغيرها من التطبيقات التى تحتوى على علاقات متعددة.

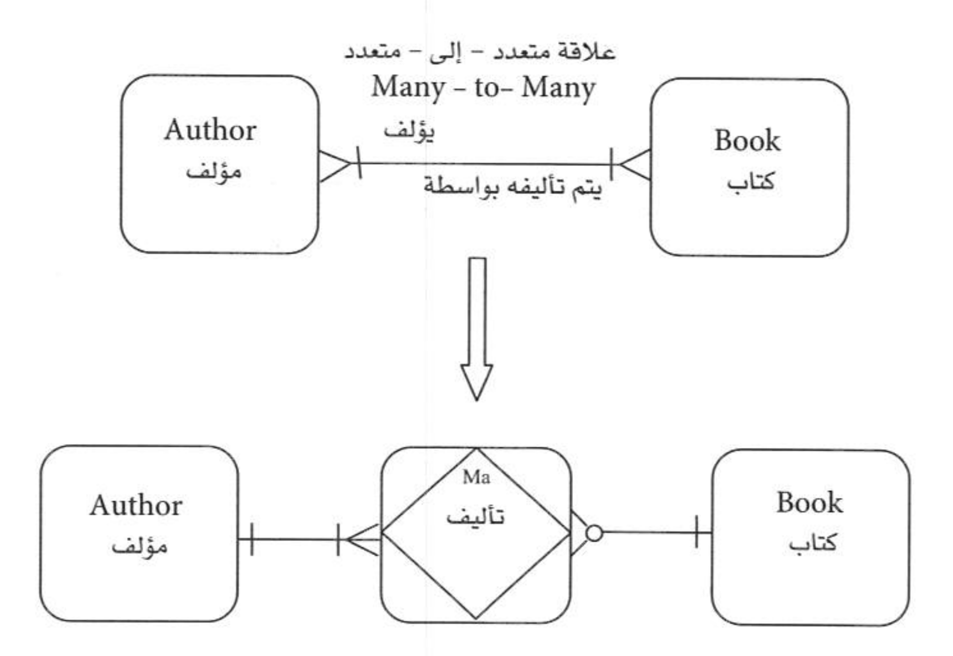


شكل (٢-١٥) علاقة متعددة وكائن الربط

يســتخدم أيضاً أسلوب كائن الربط لحل مشــكلة العلاقة المعقدة «متعدد - إلى - متعدد» الموجودة بين كائنين، وتحويلها إلى علاقتين بســيطتين من النوع «واحد - إلى - متعدد» من الناحيتين، كما هو موضح بالشكل (٢-١٦) الذى يمثل العلاقة بين المؤلف والكتاب. فطبيعة العلاقة أن المؤلف يمكن أن يؤلف كتاباً أو أكثر، وبطبيعة الحال فإن الكتاب الواحد يمكن أن يشارك فى تأليفه مؤلف واحد أو أكثر. ولتبسيط تلك العلاقة المركبة، يتم تمثيل العلاقة فى شكل كائن ربط «تأليف» يحتوى على بيانات الكتب التى شارك فى تأليفها كل مؤلف، أو بمعنى آخر بيانات المؤلفين لكل كتاب. فنجد أن كائن الربط «تأليف» يحتوى على الخصائص التالية كبيانات أساسية للكائن:

- رقم المؤلف.
- رقم الكتاب.

بالإضافــة إلــى أى بيانات أخرى تخص عملية التأليف. ويكون المفتاح الأساســى للكائن «تأليف» هو مفتاحاً مركباً يتكون من كلتا الخاصيتين.

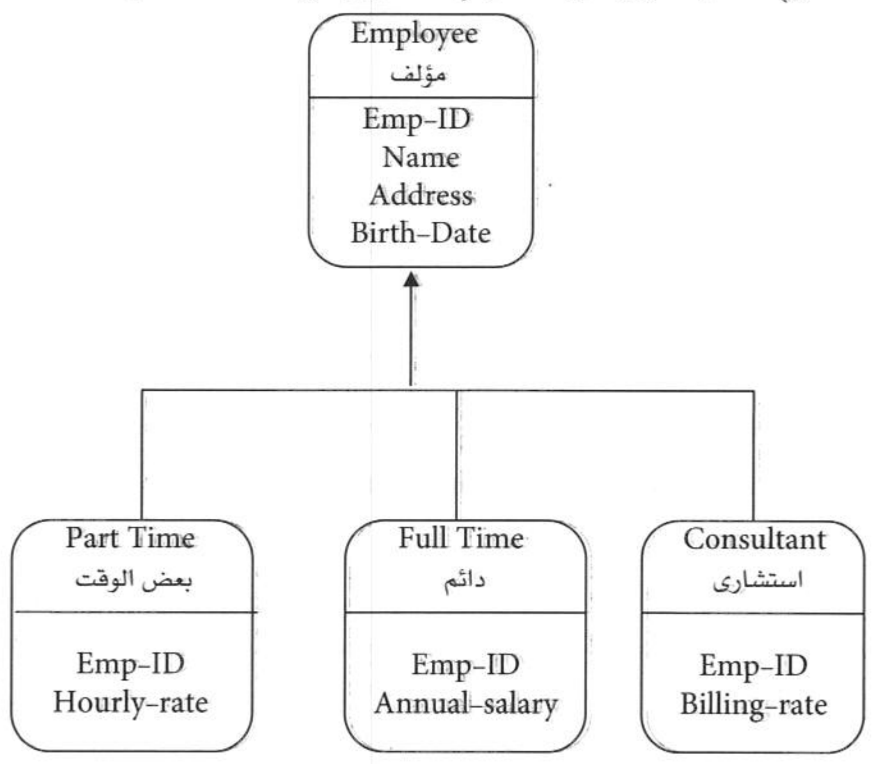


شكل (٢-١٦) استخدام كائن الربط Associative Entity

٤- التعميم Generalization:

التعميم هو أسلوب يتم فيه تجميع الخصائص المشتركة لعدد من الكائنات فى شكل نوع كائن منفصل يسمى نوع علوى Supertype. وتكون باقى الكائنات عبارة عن أنواع فرعيــة Subtype ترث الخصائص المشــتركة من النوع العلــوى Supertype وتضيف إليه خصائص أخرى تميز حالات كل نوع فرعى. ومثال ذلك يوضح الشــكل (٢-١٧) تمثيل أنواع التوظيف المختلفة فى بعض المؤسسات. فنجد أنه بغض النظر عن طبيعة الوظيفــة فإن لكل موظف مجموعة من الخصائص المشــتركة التى لا تعتمد على نوع الوظيفة مثل رقم الموظف والاســم والعنوان وتاريــخ الميلاد. وهناك خصائص أخرى تعتمــد على نوع الوظيفة مثل الراتب أو الأجر بالســاعة أو قيمة المكافأة. وعلى ذلك فقد تم تمثيل البيانات المشــتركة لجميع الموظفين من خلال كائن عام (علوى) باســم

«موظف Employee“. وتم تمثيل باقى الخصائص التى تمثل الأجر من خلال ثلاثة كائنات فرعية مرتبطة بالكائن العلوى «موظف». يحتوى كل كائن فرعى على البيانات التى تميزه بالإضافة إلى مفتاح الربط «رقم الموظف». فنجد أن الكائن «بعض الوقت Part Time“ يحتوى على بيان معدل الأجر بالساعة، فى حين يحتوى الكائن «دائم Full Time“ على بيان الراتب السنوى، ويحتوى الكائن «استشارى Consultant“ على بيان قيمة الاستشارة. ويمكن لكل كائن فرعى أن يرث (يستخدم) جميع بيانات (خصائص) الكائن العلوى «موظف»، بالإضافة إلى البيانات الخاصة به.



شكل (٢-١٧) استخدام أسلوب التعميم

٢-٢ النمذجة الموجهة بالكائنات Object-Oriented Modeling:

بعد أن استعرضنا الأسلوب التقليدى لنمذجة النظم، نقدم الآن عرضاً سريعاً للأسلوب الأكثر شيوعاً الآن فى تطوير النظم وهو أسلوب الكائنات الموجهة Object-Orientation. يعتمد هذا الأسلوب على مفهوم الكائن Object كمركب أساسى

لتطويــر النظم. يتميز الكائن بتركيب خاص يقوم علــى تغليف كل من بيانات Data وســلوك Behavior الكائن فى إطار واحد مما أدى إلى إمكانية استخدامه فى نمذجة كل مــن بيانات وعمليات النظام فى آن واحد خلال مراحل التحليل والتصميم، وكذلك يتــم اســتخدامه فى مرحلــة البرمجة من خلال لغــات البرمجة الموجهــة بالكائنات Object-Oriented Programming. يدعم هذا الأســلوب مجموعة من الخصائص المهمة والمفيدة فى بناء النظم الكبيــرة والمعقدة مثل التوريث Inheritance، والتجريد Abstraction، والتغليف Encapsulation، وتعدد الأوجه Polymorphism. بالإضافة إلى الميزة الأساسية التى يتميز بها هذا الأسلوب وهى إمكانية إعادة استخدام الكائنات المصممة جيداً فى عدة تطبيقات مشابهة، مما يوفر الوقت والمجهود والتكلفة أيضاً فى أثناء تطوير أنظمة جديدة.

الكائن Object:

الكائن هو أى شىء حقيقى له مواصفات محددة ويؤدى مهمة معينة أى أنه له دور فعلــى فى الحياة العملية. ولهذا نقول إن الكائن موجود حولنا فى كل مكان. الكائن يمكن أن يمثل شيئاً مادياً محسوساً أو مرئياً مثل الأشخاص أو الأشياء المادية (كتاب، جهاز، ســلعة، وثيقة، ..) ويمكن أيضاً أن يمثل الكائن شــيئاً غيــر مادى مثل مفهوم أو حدث (تســجيل طالــب فى دورة، حجز فى رحلة طيران، حســاب عميل فى بنك، ميزانية، ..).

والكائن له مجموعة من الخصائص أو الســمات Attributes، وله وظيفة أو سلوك Behavior. يتم وصف ســلوك الكائن عن طريق مجموعة من العمليات Operations، وهى تصف طريقة تفاعل الكائن مع الكائنات الأخرى. فمثلاً يحتوى الكائن الذى يمثل شخصاً على بعض الخصائص مثل:

الاسم، النوع، تاريخ الميلاد، العنوان، ...

ويحتوى الكائن على عمليات تصف سلوكه مثل:

يقرأ ( )، يكتب ( )، يأكل ( )، ....

وجدير بالذكر هنا أن العمليات الخاصة بالكائن هى فقط التى تستطيع أن تتفاعل مــع بياناته (خصائصه) مباشــرة، وهى بذلك تحمى الكائن مــن التعامل الخطر غير المتوقع من خارجه.

التجريد Abstraction:

المقصود بالتجريد هو التركيز على السمات (الخصائص والعمليات) التى نحتاج إليها الآن فـى أثناء عملية النمذجة وتتعلق بوجهة النظر التى يتم فحص النظام من خلالها وترك باقى السمات التى لا تتعلق بها. يختلف التجريد من مرحلة إلى أخرى فى أثناء نمذجة النظام، فعلى حسـب الاحتياج يكون النمـوذج موجزاً أو يحتوى على تفاصيل. وتكون النماذج غالباً فى المستويات الأعلى موجزة، وتكون النماذج فى المستويات الأدنى محتوية على تفاصيل أكثر، حتى نصل إلى أدنى مستوى من التجريد.

التوريث Inheritance:

يصف التوريث علاقات المشـاركة بين بعض كائنات النظام. فعندما يشـترك عدد من كائنات النظام فى بعض السمات (خصائص وعمليات) وتختلف فى البعض الآخر، يمكننا هنا تنظيمها فى شـكل صنـف عام أو علـوى Super Class، وأصناف فرعية Subclasses. يحتوى الصنف العلوى على السـمات المشتركة بينها، وتحتوى الأصناف الفرعية على السـمات الخاصة بكل كائـن. ويتم تمثيل علاقة التوريث أو الوراثة بين الصنـف العلوى والأصناف الفرعية والتى تعنى أن جميع الأصناف الفرعية يمكنها أن ترث سمات الصنف العلوى، بالإضافة إلى السمات الخاصة بها. وقد تم توضيح ذلك فى أثناء عرض علاقة التعميم Generalization بين الكائنات من خلال الشكل (٢-١٧).

تعدد الأوجه Polymorphism:

يعنى تعدد الأوجه أنه يمكن لأصناف مختلفة أو مكونات مختلفة أن تؤدى نفسـها المهمة ولكن بطريقة مختلفة. فمن المعروف فى مجال البرمجيات أنه يوجد عديد من الأدوات البرمجية التى تقوم بالوظيفة نفسها ولكن بأسلوب مختلف مثل أدوات الرسم وكتابـة النصوص ولغات البرمجة. ومـن جانب آخر يمكن أن تحتوى أصناف مختلفة على عملية بالاسـم نفسه ولكن تختلف عن بعضها فى الأسلوب أو حتى فى الوظيفة. فمثـلاً يمكن أن يتضمن عدد من الأصناف عملية باسـم «افتـح» ويكون مفهومها فى أحد الأصناف يعنى: «فتح جهاز»، وفى صنف آخر يعنى «فتح ملف»، وفى صنف ثالث يعنى «فتح حسـاب مالى»، وهكذا. ومن الواضح أنه فى كل حالة سـيتم تنفيذ عملية مختلفة تماماً مع أنها تأخذ نفس الاسـم. وتكمن أهميـة هذا المفهوم فى أنه يمكننا مـن المحافظـة على المصطلحات الطبيعيـة فى كل مجال دون اللجوء إلى اسـتخدام مصطلحات مصطنعة للتغلب على مشكلة تشابه الأسماء مع الاختلاف فى المفهوم.

التغليف Encapsulation:

يقصد بالتغليف هنا أنه يتم إخفاء تفاصيل سلوك الكائن عن الكائنات الأخرى التى تتعامــل معه. فمثلاً يتكون جهاز الحاســب الآلى من عــدة مكونات أو كائنات تتفاعل معاً لتحقيق أهداف استخدام الحاسب. ولكن ليس من الضرورى إطلاقاً أن يكون أى مكــون من الجهاز على دراية تامة بكيفية تنفيذ مكون آخر لوظيفته، ولكنه يتعامل معه ويســتخدم ما يقوم به من وظيفة دون الحاجة لمعرفة كيفية تنفيذها. فمثلاً يستخدم المعالج CPU كلاً من لوحة الإدخال، والشاشــة، والقــرص الصلب لتنفيذ مهام معينة يحتــاج إليها عند تنفيذ المهــام المطلوبة منه، ولكن لا يتطلب ذلك معرفته الكيفية التى يقوم كل مكون بأداء وظيفته. نستفيد من ذلك أنه عند حدوث خلل أو عطب فى أحد المكونات، يتم صيانته أو اســتبداله دون التأثير فى المكونات الأخرى التى تتعامل معه. هناك ميزة أخرى لمفهوم التغليف وهو أن عملية إخفاء خصائص الكائن Information Hiding عن باقى الكائنات تؤدى إلى حماية تلك الخصائص من التعامل غير الصحيح من الخارج الذى قد يعرضها للخطر أو التدمير. ولكن الصحيح أن يتم حمايتها وذلك بقصر التعامل معها من خلال عمليات الكائن نفســه فقط ويتم تصميمها بعناية تؤدى الغرض المطلوب دون تعريض البيانات للخطر.

الرسائل Messages:

تتفاعل الكائنات فيما بينها عن طريق أسلوب الرسائل، فيقوم الكائن المرسل بإرسال رســالة لكائن آخــر يحدد فيها الخدمــة المطلوبة. وبعد ذلك يقوم الكائن المســتقبل باســتقبال الرســالة، ثم يقوم بتنفيذ العمليــة المطلوبة. وتجدر الإشــارة هنا إلى أن الرســالة يجب أن تحتوى على اسم الكائن المستقبل، ثم اسم العملية المطلوب تنفيذها مصحوبة بالمعلومات الضرورية لتنفيذ العملية.

٢-٢-١ مقدمة عن لغة UML:

لقد أثبت أسلوب البرمجة المبنية على الكائنات نجاحاً مبهراً فى تطوير التطبيقات فى الفترة من أوائل التســعينيات إلى الآن، مما أدى إلى اســتخدامه على نطاق واسع واعتماده أســلوباً برمجياً قياســياً لتطويــر تطبيقات النظم. فــكان من الطبيعى أن تواكب أســاليب تحليل وتصميم النظم هذا الأســلوب فى تطوير النظم. ويتطلب ذلك استحداث طرق جديدة لنمذجة النظم تعتمد على أسلوب الكائنات الموجهة والاستفادة

من الخصائص الجديدة التى تميزه فى أثناء عمليات التحليل والتصميم. وقد ظهرت عدة محاولات جادة فى هذا المجال فى أوائل التسعينيات من جانب كل من Booch، Rumbaugh، Jacobson. وفى منتصف التسعينيات تم تجميع هذه المحاولات وتقديم منهجية موحدة لنمذجة النظم أطلق عليها اسم «لغة النمذجة الموحدة "Unified Modeling Language (UML)".

تعتبر لغة النمذجة الموحدة UML إحدى الوسائل المرئية التى تستخدم لنمذجة النظم System Modeling من خلال مجموعة من المخططات Diagrams التى تصف تركيب Structure وسلوك Behavior أى نظام معلومات. تمكن لغة UML مطورى النظم من وصف رؤيتهم لنظام برمجى معين بطريقة قياسية سهلة الفهم، وتساعدهم على تبادل وجهات النظر ومناقشتها مع مستخدمى النظام بطريقة فعالة.

تتضمن لغة UML عدة مفاهيم ورموز وإرشادات مفيدة جداً تمكن مطورى النظم من وصف النظام بدقة ومن وجهات نظر متعددة. تقوم لغة UML بتمثيل تلك المفاهيم من خلال مجموعة من المكونات والمخططات التى تمكننا من وصف النظام من منظورات Views مختلفة مثل (Rumbaugh et. Al, 2005):

- المنظور الساكن Static View أو التركيبى الذى يصف مكونات أو هيكل النظام.
- المنظور الديناميكى Dynamic View الذى يصف سلوك Behavior النظام.
- المنظور البيئى Environmental View الذى يصف البيئة المحيطة بالنظام وتفاعلها معه.
- منظور تنظيمى Organizational لتنظيم النماذج Models فى شكل حزم Packages تمكن فريق العمل من تقسيم النظم الكبيرة إلى مجموعة من النظم الفرعية مع توضيح التفاعلات بينها.

تتميز لغة UML بأنها تقوم بوصف النظام من خلال عدة نماذج (مخططات)، إذ يقوم كل نموذج بوصف النظام أو جزء منه من منظور معين، مما يؤدى إلى فهم النظام بشكل جيد، ويسهل أيضاً من عملية الاتصال والتفاهم بين أعضاء فرق التطوير بالنظام، بالإضافة إلى التفاهم والاتصال مع مجموعة مستخدمى النظام.

وعلى هذا فإنه يمكننا تعريف لغة UML بأنها لغة نمذجة متعددة الأغراض General Purpose Modeling Language. ومع أنها ليست لغة برمجة إلا أنه يمكن استخدامها من خلال إحدى أدوات هندسة البرمجيات CASE Tools لإنتاج برامج

بلغات برمجة عديدة لتنفيذ تطبيقات النظم المختلفة مثل أداة «مصمم أوراكل Oracle Designer" المقدمة من شـــركة أوراكل العالميـــة فى مجال تطوير تطبيقات نظم قواعد البيانات.

٢-٢-٢ نمذجة النظم باستخدام لغة UML:

تعتبر نمذجة النظم من الســمات الرئيسية لأغلب منهجيات تطوير النظم. ولذلك فإنــه لتطويــر نظام معلومــات بطريقة فعالة، يجــب أن يتم فحــص النظام من عدة منظورات، ومن ثم يتم نمذجة الســمات التى تميز النظــام. وكلما كان النظام كبيراً، زادت أهمية نمذجته.

يعتبر النموذج Model بمنزلة مخطط Blueprint يمثل النظام أو جزءاً منه مركزاً علــى وجهــة نظر معينة. يمكن أن يكــون النموذج موجزاً أو مفصلاً حســب الحاجة أو حســب المنظــور المطلوب تمثيلــه. ولذلك فإن نظم المعلومات تحتاج إلى أســلوب النمذجــة الــذى يمكننا من وصف النظام من عدة وجهات نظــر مختلفة تدعم عملية الفهم والتواصل بين المشاركين فى النظام. بالإضافة إلى أن نماذج النظام تعتبر جزءاً أساسياً من توثيق النظام الذى يساعد أى عضو جديد فى فريق العمل على فهم النظام والتعرف على خصائصه ومكوناته.

تعتبــر لغة النمذجة الموحدة UML هى الأســلوب القياســى المعتمد لنمذجة نظم المعلومات. يحتوى النموذج على جزأيين أساسيين هما: (Schmuller, 2004).

- رموز رسومية Notations: تمثل مكونات النماذج رسومياً والعلاقات بينها.
- معلومات دلالية Semantic Info: معلومات تصف المعنى أو المفهوم من النموذج.

تستخدم النماذج لعدة أغراض منها:

١- توصيــف متطلبات النظــام بطريقة دقيقــة يفهمها جميع المشــاركين فى النظام لدراستها والموافقة عليها.

٢- مساعدة المطورين فى عرض التصميمات المقترحة كحلول للنظام واختيار الأنسب منها قبل عملية التنفيذ.

٣- توليد مكونات تشــغيلية للنظام قابلة للاســتخدام مثل هيــاكل الكائنات ومحتوى الإجراءات وقواعد البيانات.

٤- تنظيم المعلومات داخل النظام من خلال عدة منظورات مثل المنظور الساكن لمكونات النظام Static View، والمنظور الديناميكى لسلوك الكائنات بالنظام Dynamic View.

٥- إمكانية السيطرة على الأنظمة الكبيرة والمعقدة، وذلك باستخدام أسلوب النمذجة على عدة مستويات.

٢-٢-٣ مكونات لغة UML:

لا يوجد هناك حد فاصل بين المكونات والمفاهيم فى لغة UML. ولهذا فانه من الأفضل أن نعرض مكونات لغة UML من خلال عدة منظورات Views. المنظور ببساطة هو نموذج أو أكثر يمثل النظام أو جزءاً منه من وجهة نظر أو رؤية معينة. يحتوى المنظور على مخطط Diagram أو أكثر لتمثيل المفهوم المقصود بشكل تخطيطى.

يمكننا تقسيم المنظورات إلى أربعة أنواع رئيسية هى (Fowler, 2004):

١- تركيبى Structural:

يصف مكونات النظام التركيبية والوظيفية والعلاقات بينها. يحتوى هذا التصنيف على عدة مخططات مثل مخطط الأصناف Classes Diagram، ومخطط حالات الاستخدام Use Cases Diagram، ومخطط المكونات Components Diagram.

٢- سلوك ديناميكى Dynamic Behavior:

يصف سلوك النظام من جانب التغيرات التى تحدث على مكونات النظام. يحتوى هذا النوع على مخططات مثل مخطط الحالة State Diagram، ومخطط النشاط Activity Diagram، ومخطط التفاعل Interaction Diagram.

٣- تخطيط مادى Physical Layout:

يصف هذا النوع المصادر التى يتم استخدامها لنشر النظام Deployment. يستخدم مخطط النشر Deployment Diagram لوصف هذا النوع.

٤- نموذج تنظيمى Management Model:

يصف هذا النوع الشـــكل التنظيمى لنماذج النظام فى شـــكل هرمى. يســـتخدم مخطط الحزمة Package Diagram ومخطط المكونات Component Diagram لوصف هذا النوع.

وسوف نستعرض الآن معظم المخططات التى تقدمها لغة النمذجة الموحدة UML.

٢-٢-٤ مخطط الأصناف Classes Diagram:

يعتبـــر مخطط الأصناف من المخططات الأساســـية التى تســـتخدم لوصف مكونات النظام فى الحالة الســـاكنة Static، أى أنه يصف النظام من المنظور البنائى Structural View أو المنظور الساكن Static View. يصف مخطط الأصناف أنواع الكائنات الموجودة بالنظام والعلاقات بينها فى حالة السكون. يصف المخطط أيضاً خصائص وعمليات كل صنف فى النظام، بالإضافة إلى القيود Constraints التى تحكم اتصال الكائنات بعضها ببعـــض. يتـــم توصيف مجموعة الكائنات المشـــتركة فى التركيب Structure والســـلوك Behavior فى شكل صنف Class يحتوى على خصائص Attributes الكائنات والعمليات Operations التـــى تصف ســـلوكها. يعتبر الصنف Class هو المكون الأساســـى لمخطط الأصناف، ويتم تمثيله من خلال صندوق يتكون من ثلاثة أجزاء هى (شكل (٢-١٨)):

- اسم الصنف Class-Name.
- الخصائص Attributes.
- العمليات Operations.

Class Name
Attribue: Type ....
Operation(): Return-Type ....

شكل (٢-١٨) الشكل العام للصنف Class

الخصائص Attributes:

تعتبر الخاصية هى إحدى السمات التى تميز صنف الكائن وتصف مجموعة القيم التى يمكن أن يأخذها الكائن. يمكن أن يحتوى الصنف على عديد من الخصائص.

الشكل العام للخاصية فى لغة UML هو:

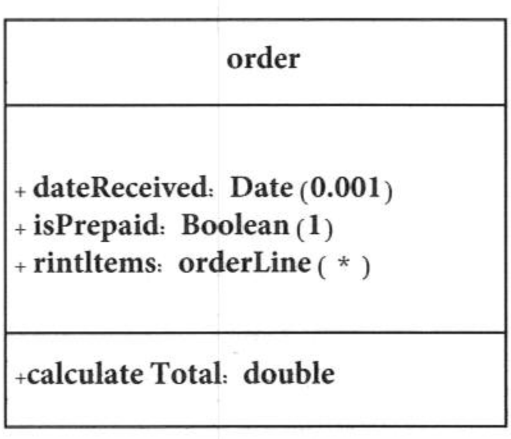
Visibility name: type multiplicity = default {property string}

حيث تحدد visibility مدى اســتخدام الخاصية، وهى إما عام Public ويرمز لها بالرمز (+)، أو خاص Private ويرمز لها بالرمز (-)، وسيتم شرحها فيما بعد.

وتحــدد multiplicity عدد الكائنات التى يمكن أن تأخذ الخاصية (صفر أو واحد أو أكثر).

وتمكننا default من إضافة قيمة افتراضية للخاصية.

وتمكننــا property string مــن إضافــة أى معلومة أخــرى للخاصية مثل read only لتوضيح عدم إمكانية التعديل فى قيمة الخاصية. يوضح شــكل (٢-١٩) مثالاً لخصائص الصنف order.



شكل (٢-١٩) خصائص وعمليات الصنف order

العمليات Operations:

تمثــل العمليات مجموعة الأفعال Actions التــى يمكن أن يقوم بها صنف الكائن. وكما ذكرنا من قبل، فإن العمليات تمثل واجهة الصنف، حيث يتم التعامل الخارجى مع

الكائن من خلالها فقط، وهى التى تقوم بدورها بالتعامل مع خصائص الكائن. هناك نوعان أساسيان من العمليات هما:

١- عمليات تقوم بتحديد قيم خصائص الصنف مع إمكانية التعديل فيها، أى أنها تغير من حالة الكائن، وتسمى Setting Methods أو Modifiers.

٢- عمليات تقوم بالاستعلام عن قيم خصائص الصنف، ولا تقوم بتغييرها، أى أنها لا تغير من حالة الكائن، وتسمى Getting Methods أو Query.

الشكل العام للعملية فى لغة UML هو:

Visibility name(parameter-list): return-type {property string}

تمثل return-type نوع القيم المسترجعة من العملية فى حالة Query.

العلاقات Relationships:

تحتوى لغة UML على مجموعة من العلاقات المختلفة التى تربط بين مكونات اللغة. يعرض شكل (٢-٢٠) مجموعة العلاقات ووصفاً مختصراً لها، بالإضافة إلى الرموز المستخدمة لتمثيل كل علاقة فى لغة UML.

العلاقة	الوصف	الرمز
الارتباط Association	تصف عملية الربط بين كائنات الأصناف المختلفة Objects.	———
التعميم Generalization	تصف العلاقة بين الأصناف العلوية Super-classes والأصناف الفرعية Sub-classes فى أثناء عملية التوريث Inheritance.	←——

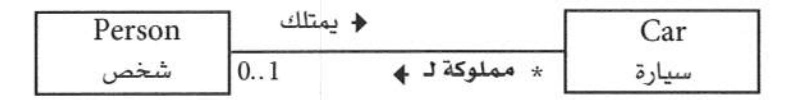
التحقيق Realization	تصف العلاقة بين صنـف الواجهة Interface والأصنـاف التـى تحققـه أو تطبـق مواصفاته السلوكية Implementation.	---------◄
الاعتمادية Dependency	تصف علاقـة الاعتمادية الموجودة بين صنفين يكون لأحدهما تأثير فى الآخر.	---------<

شكل (٢-٢٠) أنواع العلاقات فى لغة UML

وفيما يلى وصف موجز لكل نوع من أنواع العلاقات بين الأصناف فى لغة UML.

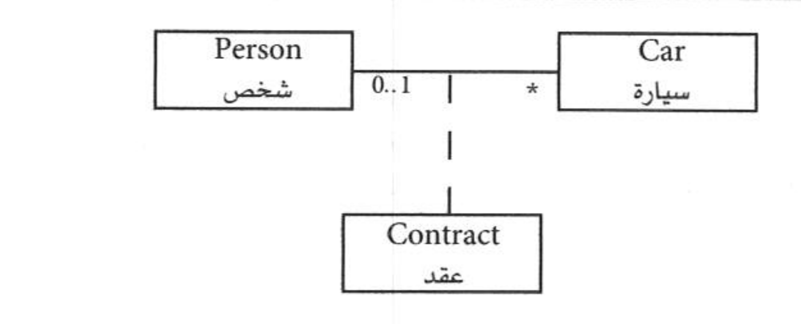
الارتباطات Associations:

يتـم توصيف الارتباط المنطقى بين أصناف الكائنات باسـتخدام خط توصيل بين الأصنـاف يحدد نوعية الارتباط. ويمكن إضافة مقطـع نصى لتوضيح نوع الارتباط كما هو موضح بشكل (٢-٢١).



شكل (٢-٢١) الارتباط بين أصناف الكائنات

أحياناً تحتوى عملية الارتباط نفسها على بعض الخصائص والعمليات، وهنا يجب تمثيل عملية الارتباط من خلال صنف خاص يسـمى صنـف الارتباط Associative Class، ويتم ربطه بالأصناف الأخرى داخل النظام. كمثال على ذلك فإنه يمكننا تمثيل عملية الارتباط بين الشـخص والسيارة فى شـكل صنف ارتباط يسمى «عقد» يصف عملية الارتباط بين الصنفين من خلال عملية تحرير عقد يحتوى على عدة خصائص منهـا تاريخ البيع والسـعر وطريقة الدفع. يوضح شـكل (٢-٢٢) كيفية تمثيل صنف الارتباط «عقد» بطريقتين مختلفتين فى لغة UML.

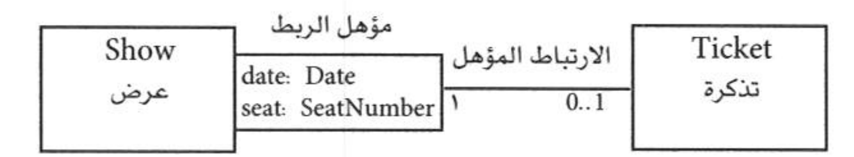




شكل (٢-٢٢) صنف الارتباط

الارتباطات المؤهلة Qualified Associations:

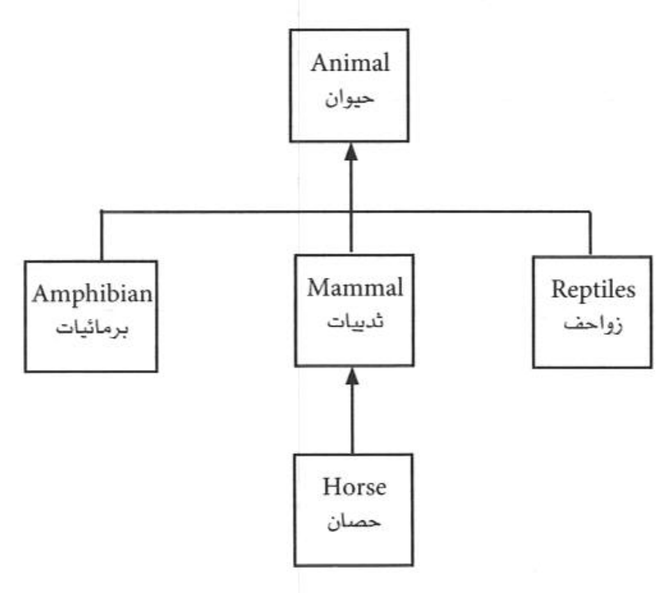
عندمـا تكون قيمة أحد الخصائص للصنف هى المؤهلـة لاختيار الكائنات المقابلة لهـا فى صنـف آخر مرتبطة به، فإن هذه الخاصية تسـمى مؤهل الربط Qualifier. وعلى هذا الأسـاس يتم اعتماد قيم هذه الخاصية لاختيار كائنات معينة من مجموعة الكائنات المرتبطة التى تحقق عملية الارتباط بين الصنفين. تستخدم لغة UML رمز المسـتطيل ملاصقاً للصنف الذى يحتوى على الخاصية المؤهلة. يوضح شكل (٢-٢٣) مثالاً على استخدام الارتباط المؤهل حيث تم اختيار مؤهل الربط المكون من خصائص «رقـم المقعـد Seat Number" و «التاريخ Date" لتحقيق عمليـة الربط بين الصنف «عـرض» والصنـف «تذكرة» مما يعنى أن كل تذكرة تكـون مخصصة لمقعد محدد فى تاريخ محدد لأحد العروض.



شكل (٢-٢٣) الارتباط المؤهل

التعميم Generalization:

تعتبر خاصية التوريث من المفاهيم الأساسية لأسلوب الكائنات الموجهة. وتكمن أهميتها فى أنها موجودة بكثرة فى معظم الأنظمة التى نتعامل معها فى حياتنا اليومية. يتم تمثيل الخصائص والعمليات المشتركة فى شكل صنف عام general class أو صنف علوى super class. ويتم تمثيل الخصائص والعمليات التى تخص كل نوع فى شكل أصناف فرعية sub classes ترتبط بالصنف العلوى من خلال علاقة التعميم. يتم تمثيل علاقة التعميم باستخدام سهم على شكل مثلث يبدأ من الأصناف الفرعية ويتجه نحو الصنف العلوى. تسمح هذه العلاقة للأصناف الفرعية أن تستخدم جميع الخصائص والعمليات الموجودة بالصنف العلوى مع إمكانية تغييرها، وذلك بالإضافة إلى الخصائص والعمليات الموجودة بكل صنف فرعى. أحياناً يطلق على الصنف العلوى اسم الصنف الأب parent class والصنف الفرعى اسم الصنف الابن child class. يمكننا تمثيل تلك العلاقة على عدة مستويات، فيمكن للصنف الابن أن يكون أباً لصنف أو أصناف أخرى فرعية. يوضح شكل (٢-٢٤) مثالاً لتمثيل خاصية التعميم فى عدة مستويات. نلاحظ فى هذا المثال أن الصنف «حصان "horse" هو نوع من الصنف "ثدييات mammal" وهو يعتبر أيضاً نوعاً من الصنف "حيوان animal".



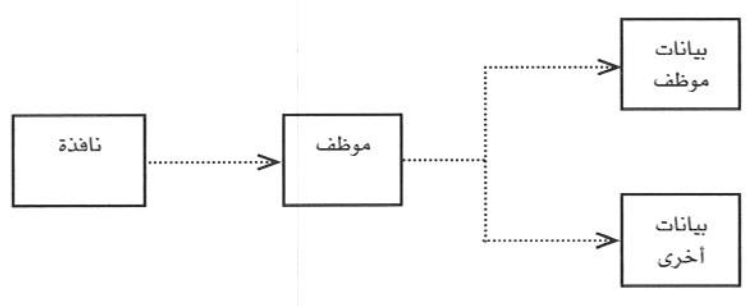
شكل (٢-٢٤) مثال لخاصية التعميم

الاعتمادية Dependency:

تعتبـــر الاعتمادية نوعاً من العلاقات بـــين الأصناف التى يقوم فيها أحد الأصناف «عميل client" باســـتخدام صنف آخر «مزود supplier"، ويعنى هذا أن أى تغيير فى الصنف supplier يؤدى إلى تغيير فى الصنف client، ولذلك فهو يعتبر معتمداً عليه. توجد الاعتمادية بين الأصناف لعدة أسباب منها:

- أن يرسل صنف رسالة إلى صنف آخر لاستخدام إحدى وظائفه.
- أن يكون صنف ما عبارة عن جزء من بيانات صنف آخر.
- أن يكون أحد الأصناف عبارة عن متغير ممرر passing parameter لإحدى عمليات صنف آخر.
- أن يكون أحد الأصناف معرفاً كمتغير محلى داخل إحدى عمليات صنف آخر.
- أن يكون أحد الأصناف معرفاً كمتغير عام Global فيصبح متاحاً لكل العمليات.

يتـــم تمثيل الاعتمادية فى لغة UML باســـتخدام خط متقطع ينتهى بســـهم موجه ناحيـــة الصنـــف supplier، كما هو موضح بالشـــكل (٢-٢٥). يتضح من الشـــكل أن الصنف «نافذة» يعتمد على الصنف «موظف» الذى بدوره يعتمد على الأصناف «بيانات موظف» وبيانات أخرى» وهى تزوده ببيانات الموظف التى يتم عرضها من خلال صنف النافذة.

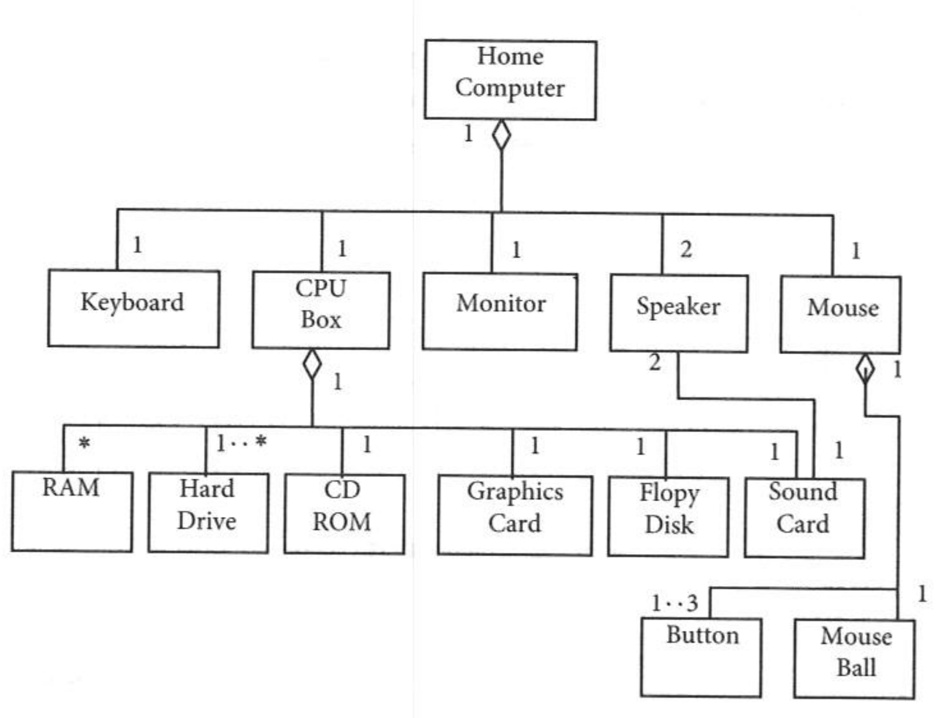


شكل (٢-٢٥) مثال للاعتمادية

التجميع Aggregation:

أحياناً يحتوى النظام على صنف عام (مجمع) وهو عبارة عن مجموعة من الأصناف الأخرى التى تم تجميعها فى شـــكل الصنف المجمع. وهنا نجد أن هناك علاقة تجمع

بين الصنف المجمع والأصناف المكونة له وهي تسمى علاقة «جزء-كل» "part-whole" أو علاقــة التجميــع Aggregation. فمثلاً يمكننا وصف العلاقة بين جهاز حاســب وأجزائه المختلفة فى شــكل علاقة تجميع. يوضح شــكل (٢-٢٦) تلك العلاقة حيث يعتبر الصنــف "Home Computer" هو الصنف المجمــع "whole"، وهو يتكون من عدة أجزاء، كل جزء فى شكل صنف "part" مثل "keyboard"، "monitor"، "mouse" "CPU-Box". ونجد أيضاً أن أحد هذه الأجزاء "CPU-Box" يتكون من عدة أجزاء، ولهــذا يعتبر صنفاً مجمعاً "whole" لأصناف أخرى مكونة له فى شــكل أصناف "part" مثــل "Hard-Drive"، "RAM"، "CR-ROM". وكذلك نجد أن الصنف "mouse" هو بمنزلــة صنف مجمع لكل مــن "Mouse-Ball"، "Button". يتم تمثيل علاقة التجميع فى لغة UML فى شكل خطوط تصل الأجزاء بالمجمع لها وتنتهى بشكل «معين مفرغ» يتجه نحو الصنف المجمع.



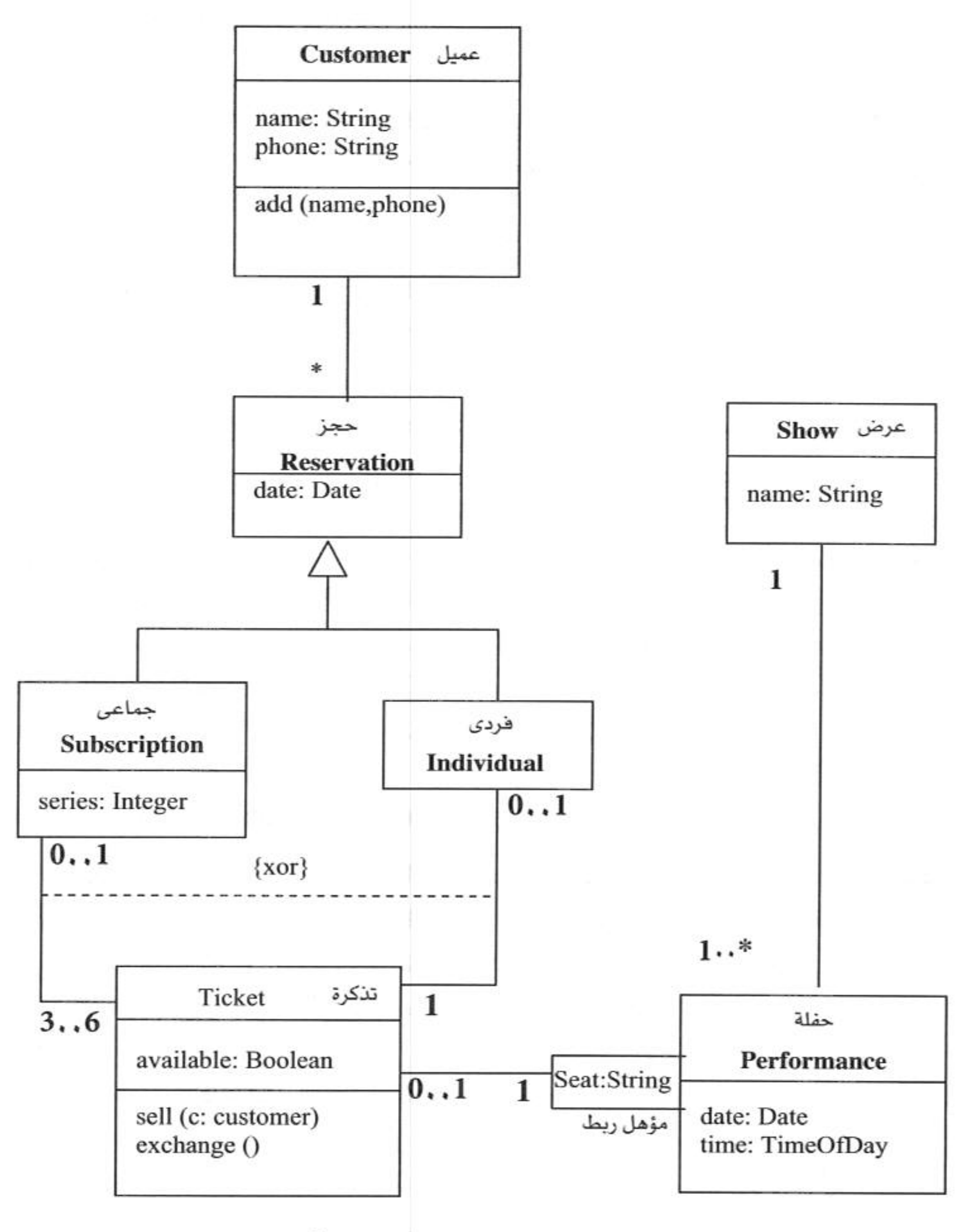
شكل (٢-٢٦) علاقة التجميع

مثال لاستخدام مخطط الأصناف:

يوضح شـــكل (٢-٢٧) مثالاً لاســـتخدام مخطط الأصناف لوصف الشكل البنائى لنظام حجز وبيع التذاكر فى إحدى دور العرض الترفيهية. يتميز هذا النظام باحتوائه علـــى معظم أنواع العلاقات بين الأصناف التـــى تم ذكرها آنفاً. يحتوى المخطط على مجموعة الأصناف الأساسية التى تمثل عملية الحجز مثل «عميل Customer" و«حجز Reservation" و «تذكرة Ticket" و «حفلة Performance"و «عرض Show". تحتوى عملية الحجز على نوعين: الأول هو الحجز الفردى لشخص واحد، والثانى هو الحجز الجماعى لمجموعة من الأشـــخاص لا تقل عن ثلاثة ولا تزيد على ستة أشخاص. يتم تنفيـــذ العـــرض الترفيهى فى أكثر مـــن حفلة فى اليوم أو فى الأســـبوع. يبدأ العميل بتحديـــد عملية الحجز (فردى/جماعـــى) ويمكنه أن يقوم بإجراء حجز واحد أو أكثر. يتم تمثيل عملية الحجز باستخدام أسلوب علاقة التعميم من خلال صنف عام «حجز» وصنفـــين فرعيين لـــه هما «فردى»، «جماعـــى» كما هو موضح بالشـــكل. وبعد ذلك يتم اســـتخراج التذاكر للعميل، وذلك بإصدار تذكرة للحجز الفردى وعدد من التذاكر يناســـب عدد الأشخاص فى الحجز الجماعى. تصدر التذكرة للعميل فى مقعد معين فـــى حفلة معينة (بتاريـــخ ووقت محددين) لعرض معين. ويتـــم تمثيل ذلك من خلال علاقة الربط المؤهـــل بين صنف «تذكرة» وصنف «حفلة» باســـتخدام خاصية «المقعد Seat" باعتبارها مؤهلاً لعملية الربط.

٢-٢-٥ مخطط حالات الاستخدام Use Cases Diagram:

يستخدم مخطط حالات الاستخدام لوصف المتطلبات الوظيفية للنظام. يتم تحديد تلـــك المتطلبات فى أثناء مرحلة التحليل من خلال مجموعة التفاعلات Interactions التى تحدث فعلياً بين مســـتخدمى النظام والنظام نفســـه. فكل مستخدم له طريقته فـــى التعامـــل مع النظام من خلال مجموعة من الإجراءات فى شـــكل ســـيناريو معين Scenario. والسيناريو هو عبارة عن مجموعة الخطوات الإجرائية التى تصف تفاعل المســـتخدم مع النظام لتنفيذ وظيفة محـــددة. يتم وصف أى إجراء وظيفى يتفاعل به المستخدم مع النظام (ســـيناريو) من خلال وحدة وظيفية تسمى «حالة استخدام Use Case". وبالطبع فإن تعامل المســـتخدمين مع النظام يمكن حصره فى شـــكل مجموعة من السيناريوهات أو حالات الاستخدام Use Cases.



شكل (٢-٢٧) مثال لمخطط الأصناف

يستخدم مخطط حالات الاستخدام لوصف تفاعل مجموعة المستخدمين مع النظام من خلال مجموعة من حالات الاستخدام. تكمن أهمية هذا المخطط أثناء عملية التحليل للنظام، حيث يتم استخدامه لوصف متطلبات النظام الوظيفية من وجهة نظر مستخدمى النظام.

الفاعل Actor:

الفاعل هو أى مستخدم للنظام يقوم بدور فعال فى أثناء التعامل معه. ويمكن أن يكون الفاعل شخصاً Person أو جهازاً Device أو أى نظام فرعى Sub-system آخر يتفاعل مع النظام الحالى. والفاعل من النوع (شخص) يمكن أن يكون عميلاً Customer أو مندوب مبيعات Salesperson أو مديراً Manager أو غيره. الفاعل من النوع (جهاز) يمكن أن يكون قارئاً آلياً لأكواد الأصناف Bar-code أو كاميرا أو خطاً هاتفياً أو غيره. الفاعل من النوع (نظام فرعى) يمكن أن يكون نظاماً آخر يتكامل مع النظام الحالى وأنظمة أخرى لتكوين نظام كبير أو نظام آخر خارجى يتطلب التعامل مع النظام الحالى.

نموذج حالة استخدام Use Case Model:

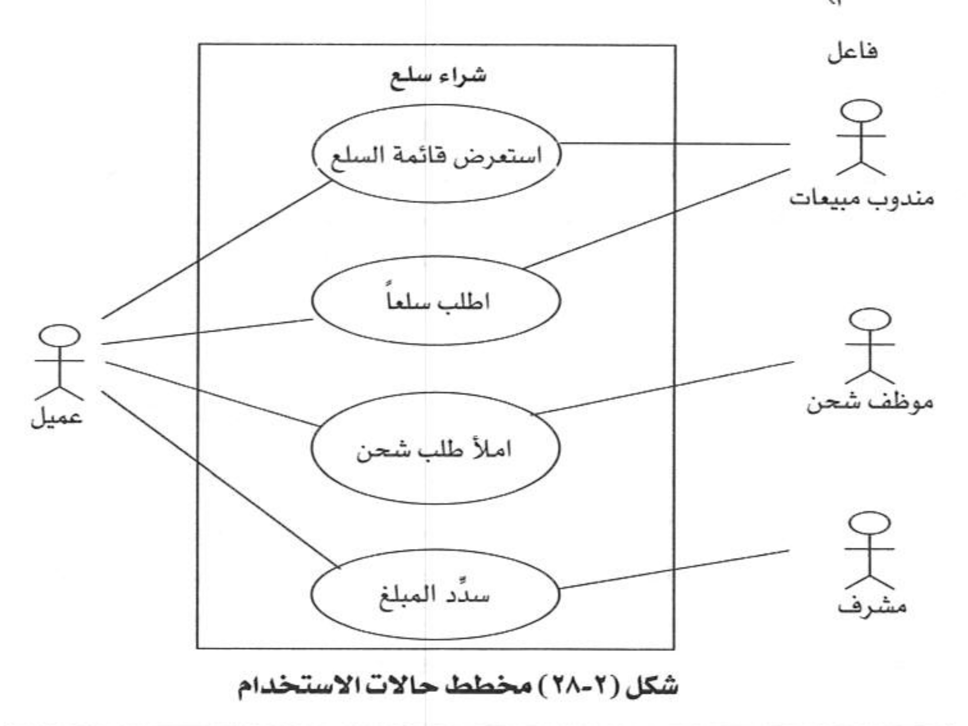
يبدأ الفاعل Actor عادة فى التعامل مع حالة الاستخدام Use Case بطريقة ما مثل إدخال بيان أو إرسال رسالة، ويمكنه أيضاً أن يستقبل مخرجات من حالة الاستخدام. يعتبر مخطط حالات الاستخدام بمنزلة شكل رسومى يوضح جميع حالات استخدام النظام من قبل مستخدمى النظام. وهو يشابه مخطط السياق Context Diagram المستخدم فى طريقة التحليل الهيكلى. فهو يصف حدود النظام وتفاعله مع المستخدمين من الخارج، وهو يحتوى على:

- الفاعلون Actors.
- حالات الاستخدام Use Cases.
- العلاقات بين حالات الاستخدام والفاعلين Relationships.

يمكننا استخدام مخطط حالات الاستخدام لوصف نظام كامل أو وصف نظام فرعى (جزء من نظام أكبر) أو لوصف أجزاء داخلية صغيرة مثل حالات استخدام صنف معين. جدير بالذكر هنا أنه إذا كان محتوى حالة الاستخدام كبيراً بحيث يحتوى على تفاصيل عديدة، فيمكننا أن نجزئها إلى مجموعة من حالات الاستخدام البسيطة.

يوضح شكل (٢-٢٨) مثالاً لاستخدام مخطط حالات الاستخدام لوصف المتطلبات الوظيفية لعملية شراء السلع. يحتوى المخطط على أربعة فاعلين «عميل» و«مندوب مبيعات» و«موظف شحن» و«مشرف». يقوم العميل باستعراض قائمة السلع بالاشتراك مع مندوب المبيعات لاختيار السلع المطلوب شراؤها من خلال حالة الاستخدام «استعرض سلعاً». بعد ذلك يقوم النظام بعرض بيانات أسعار السلع المختارة وتكاليف شحنها، ومن ثم يقوم العميل بتعديل القائمة (بالحذف أو الإضافة) ثم يقوم بتأكيد عملية الطلب من خلال حالة الاستخدام «اطلب سلعاً». بعد ذلك يقوم موظف الشحن بالاشتراك مع العميل بملء بيانات طلب الشحن من خلال حالة الاستخدام «املأ طلب شحن». ثم يقوم العميل بالاشتراك مع المشرف بإدخال بيانات التسديد لفاتورة شحن السلع المطلوبة مثل بيانات الدفع الفورى أو بيانات الدفع بشيك مصرفى أو بيانات التسديد الإلكترونى باستخدام بطاقات الائتمان.

يتم الاستفادة أيضاً من مخطط حالات الاستخدام فى تحديد الأصناف المستخدمة بالنظام Classes، وتحديد العمليات لكل صنف (باستخدام محتوى حالة الاستخدام).



شكل (٢-٢٨) مخطط حالات الاستخدام

محتوى حالة الاستخدام Use Case Content:

كمـا ذكرنا مـن قبل فإن حالة الاسـتخدام تمثل أحد سـيناريوهات التفاعل بين المستخدم والنظام، وأن السيناريو يتكون من مجموعة من الخطوات الإجرائية لتنفيذ مهمة أو وظيفة معينة. وكما لاحظنا فإن تلك الخطوات لا تظهر فى مخطط حالات الاسـتخدام، وإنمـا تظهر الحالات فقط وتفاعلها مع المسـتخدمين. لأن مخططات حالات الاسـتخدام تعتبر جزءاً مهماً من وثائـق تحليل وتصميم النظام، ويتم الرجوع إليهـا دائمـاً من قبل فريق التطوير بالنظام لاسـتخراج مكونـات مخطط الأصناف وتحديد الخصائص والعمليات المطلوبة وكذلك أثناء تحديد حالات الاختبار للوحدات الوظيفيـة، فإنه يتحتم علينا إعـداد محتوى جميع حالات الاسـتخدام الموجودة فى المخططـات وتضمينها مع وثائق النظام لتتم عملية الفهم لفريق التطوير. لم تفرض لغة UML شـكلاً محـدداً لوصف محتوى حالات الاسـتخدام ولكنه من الأفضل أن تحتوى على ما يلى:

- اسم حالة الاستخدام، ووصف مختصر لها.
- اسم الفاعل الأساسى الذى يبدأ تنفيذ حالة الاستخدام.
- مجموعة الافتراضات Assumptions (إن وجدت).
- الشروط المسبقة لحالة الاستخدام Preconditions.
- خطوات تنفيذ حالة الاستخدام Steps.
- الشروط اللاحقة بعد انتهاء السيناريو Postconditions.
- اسم الفاعل المستفيد من مخرجات حالة الاستخدام.
- اسـتخدام الامتـدادات Extensions فـى حالة وجود شـرط يؤدى إلـى إجراءات مختلفة.

يوضح شـكل (٢-٢٩) محتوى حالة الاستخدام «شراء سـلع» الموضحة بالشكل (٢-٢٨).

اسم حالة الاستخدام: شراء سلع
الفاعل الأساسى: عميل
السيناريو الأساسى:
١- يستعرض العميل قائمة السلع المعروضة.
٢- يختار العميل السلع التي يريد شراءها.
٣- يعرض النظام بيانات أسعار السلع المختارة وتكلفة الشحن.
٤- يقوم العميل بتعبئة بيانات شحن السلع المختارة.
٥- يقوم العميل بإدخال بيانات الدفع لفاتورة الشحن.
٦- يقوم النظام بالتحقق من بيانات الدفع.
٧- يؤكد النظام على إتمام عملية البيع.
٨- يرسل النظام رسالة بالبريد الإلكترونى للعميل لتأكيد عملية البيع.

امتدادات:
- فى حالة فشل عملية التحقق من بيانات بطاقة الدفع:
- يقوم العميل بإعادة إدخال بيانات بطاقة الدفع،
أو يقوم العميل بإلغاء العملية.

الفاعل المستفيد: عميل

شكل (٢-٢٩) مثال لمحتوى حالة استخدام

العلاقات بين حالات الاستخدام Relationships Among Use Cases:

تتفاعل حالات الاستخدام داخل المخطط بعضها مع بعض وكذلك مع الفاعلين بعدة أشكال. يتم تمثيل أشكال التفاعلات بين حالات الاستخدام ومع الفاعلين من خلال مجموعة من العلاقات مثل «الارتباط Association و «التعميم Generalization" و «التضمين Inclusion" و «الامتداد Extension" كما هو موضح بالشكل (٢-٣٠). تنشأ علاقة الارتباط طبيعياً بين الفاعل وحالات الاستخدام للنظام. أما باقى العلاقات (وهى بين حالات الاستخدام فقط) فتكون نتيجة لعملية إعادة تنظيم مكونات حالات الاستخدام الأولية لتلائم خصائص ومميزات أسلوب الكائنات الموجهة الذى يتم اتباعه فى أثناء عمليات تحليل ونمذجة متطلبات النظام. فإذا كانت هناك مجموعة من

حالات اســتخدام النظام تشترك فى بعض المتطلبات، فإنه يمكننا أن نستخدم علاقة التعميم لفصل الجزء المشترك فى شكل حالة استخدام عامة، وترتبط بعلاقة التعميم مع حالات الاستخدام الفرعية والمكملة لها حسب كل متطلب. وبالمثل يمكننا استخدام علاقة التضمين بين حالة استخدام معينة وحالات استخدام أخرى تكون معدة مسبقاً لتكون ضمن مكونات حالة الاســتخدام الحالية، مما يدعم أســلوب إعادة الاستخدام Reusability الذى يتميز به أسلوب الكائنات الموجهة. وكذلك يمكننا استخدام علاقة الامتداد بين حالة اســتخدام معينة وحالات اســتخدام أخرى جديدة تنشــأ فى أثناء عمليات التحديث والصيانة للنظام، وذلك بدون التعديل فى مكونات حالة الاســتخدام الحالية.

العلاقة	الوصف	الرمز
الارتباط Association	تصــف عملية الربط بين حالات الاســتخدام والفاعلين Actors.	――――――
التعميم Generalization	تصــف العلاقــة بين حالــة اســتخدام عامة Parent وحــالات اســتخدام فرعية Child تــرث خصائص الحالة العامــة وتضيف إليها Inheritance.	◁――――――
التضمين Inclusion	إمكانية اســتخدام حالات استخدام أخرى من داخل حالة استخدام معينة.	«Include» <----------
الامتداد Extension	إمكانية إضافة حالات اســتخدام جديدة إلى حالة استخدام معينة.	«Extend» <----------

شكل (٢-٣٠) العلاقات بين حالات الاستخدام

وفيما يلى وصف مختصر لكل نوع من أنواع العلاقات فى مخطط حالات الاستخدام.

التضمين Inclusion:

يتطلب أحياناً أن تتضمن إحدى حالات الاستخدام سلوك behavior حالات استخدام أخرى كجزء من ســلوكها. وهنا يتم ربط حالة الاستخدام بالحالات المتضمنة للسلوك

المطلوب من خلال علاقة تسـمى «تضمين Inclusion>>. يتم تمثيل تلك العلاقة فى لغـة UML مثل علاقة الاعتمادية بين الأصناف، وهــى عبارة عن خط متقطع ينتهى بسـهم يشـير ناحية الحالات المتضمنة، ويكتب بجواره كلمــة <<Include>>، كما فى الشكل (٢-٣١).

الامتداد Extension:

يتطلـب فى أحيـان أخرى أن تكـون حالة اسـتخدام معينة هى امتـداد إضافى Incremental Extension لحالة اسـتخدام أساسـية Base Use Case. ومن الممكن أن تكون هناك أكثر من حالة استخدام امتداداً لحالة الاستخدام الأساسية. يتم ربط حالات الاستخدام (الامتداد) بحالة الاستخدام (الأساسية) من خلال علاقة الامتداد Extension. يتـم تمثيل تلك العلاقة فى لغة UML مثل علاقة التضمين، ولكن يكون اتجاه السـهم ناحية حالة الاستخدام الأساسـية، ويكتب جوارها كلمة <<Extend>>، كما فى شكل (٢-٣١).

التعميم Generalization:

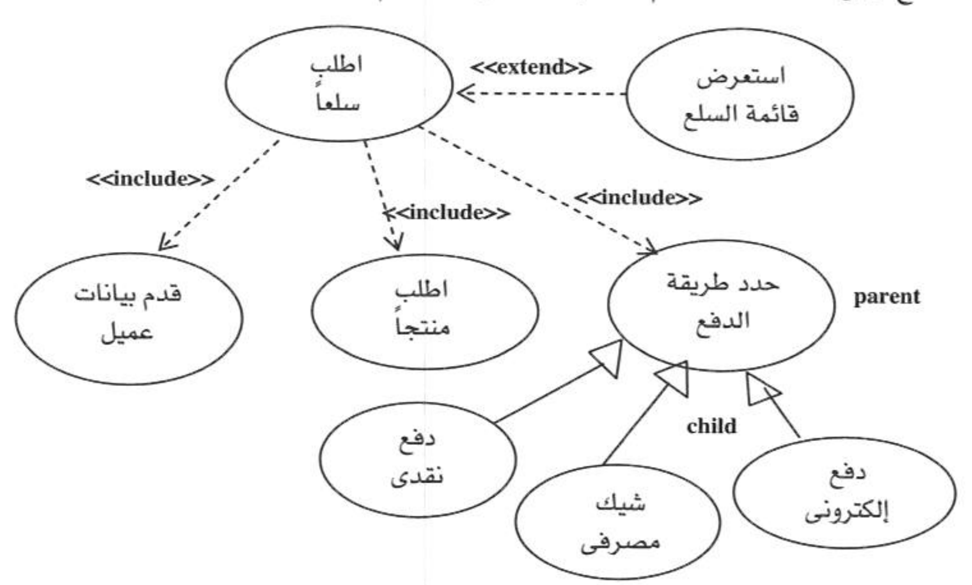
يمكننا تطبيـق مفهوم التوريث Inheritance بين حالات الاسـتخدام أيضاً. فمن الممكن تصميم حالات الاستخدام المشتركة فى بعض السلوك فى شكل حالة استخدام عامــة أو أب Parent Use Case ومجموعة من حالات الاسـتخدام الفرعية أو الأبناء Child use Cases. وهنا يمكن للحالات الفرعية أن ترث سلوك الحالة العامة وتضيف إليها سلوكها الخاص بها. يتم تمثيل تلك العلاقة كما هى تماماً بين الأصناف وتسمى أيضاً علاقة التعميم Generalization بين حالات الاسـتخدام. ويشـير المثلث (رأس العلاقة) إلى الحالة العامة، كما فى شكل (٢-٣١).

التجميع Grouping:

عندما يكون عدد حالات الاسـتخدام كبيراً، فمن الأفضل تنظيم المخطط فى شكل مجموعـات. وكل مجموعة تحتـوى على عدد من حالات الاسـتخدام المتفاعلة لأداء غـرض معين. يحدث ذلـك غالباً عندما يكون النظام مكوناً من مجموعة من الأنظمة الفرعية أو يمكن تجزئته إلى مجموعة من المكونات الوظيفية. ونحتاج هذه الخاصية دائماً فى أثناء مرحلة توصيف متطلبات النظام، حيث يتم وصف كل متطلب فى شكل

مخطط حالات اســتخدام منفصل، ثم يتم تجميع تلك المخططات فى شــكل مخطط مجمع للنظام.

يوضح شكل (٢-٣١) أسلوباً آخر لوصف مخطط حالات الاستخدام لعملية «شراء ســلع» المقدمة فى الشــكل (٢-٢٨) وذلك بعد إعادة تنظيم حالات الاستخدام لتلائم أســلوب إعادة الاســتخدام لبعض المتطلبات الوظيفية للنظام. فقد تم فصل عمليات طلب المنتج وتقديم بيانات العميل من حالة الاســتخدام «اطلب ســلعاً» ووضعهما فى شــكل حالات اســتخدام منفصلة «اطلب منتجاً» و «قدم بيانــات عميل»، وتم ربطهما بحالة الاســتخدام الأصلية «اطلب ســلعاً» من خلال علاقــة التضمين، وذلك لإتاحة إمكانية استخدامهما من قبل حالات استخدام أخرى داخل النظام أو خارجه. وكذلك تم إعــادة تنظيم عملية الدفع من خلال حالة اســتخدام عامة «حــدد طريقة الدفع» ويرتبط بها ثلاث حالات فرعية لثلاثة أنواع لتســديد فاتورة شــحن السلع وهى «دفع نقدى» و«دفع إلكترونى» و«شيك مصرفى». وقد تم ربط حالة الاستخدام العامة «حدد طريقة الدفع» بحالة الاستخدام «اطلب سلعاً» من خلال علاقة التضمين، إذ إنها تعتبر جزءاً من عملية طلب الســلع. وقد تم أيضاً إضافة حالة الاستخدام «استعرض قائمة السلع» إلى حالة الاستخدام «اطلب سلعاً» باستخدام علاقة الامتداد.



شكل (٢-٣١) العلاقات فى مخطط حالات الاستخدام

٢-٢-٦ مخطط التتابع Sequence Diagram:

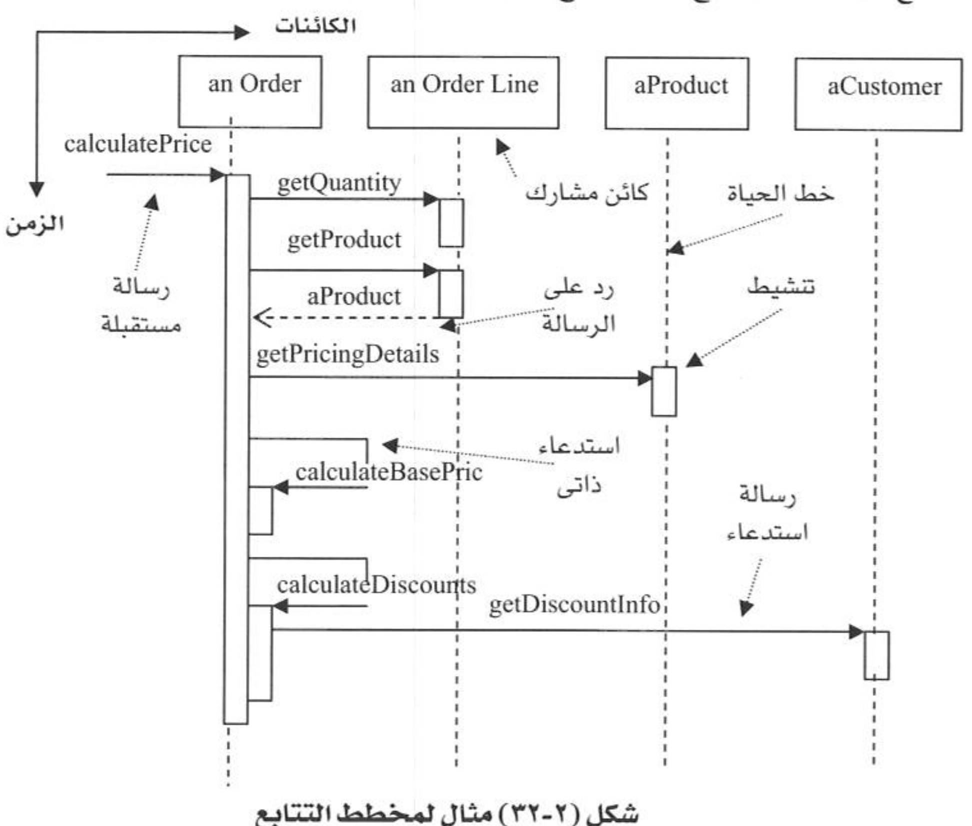
يمكننا التعرف على سـلوك النظام من خلال التفاعل الذى يتم بين كائنات النظام. هنـاك طريقتان لوصف هـذا التفاعل. يتم التركيز فى الطريقة الأولى على سـلوك كائن واحد، وهو ما سيتم تقديمه من خلال مخطط الحالة. وتتضمن الطريقة الثانية مجموعة من الكائنات المتفاعلة التى تمثل وجهة نظر مفصلة لسلوك النظام أو جزء منه من خلال سـيناريو معين. تحدث التفاعلات بين الكائنات فى شكل خطوات متتالية، وكل خطوة تسـتغرق فترة زمنية محددة. وعلى هذا، فإنه لا بد من تحديد السـيناريو أولاً، ثم نبدأ بتمثيل التفاعلات بين الكائنات طبقاً لهذا السيناريو. تستخدم لغة UML مخطط التتابـع Sequence Diagram لتمثيل هذا النوع من التفاعلات. يتم التفاعل بين الكائنات من خلال الرسائل الممررة بينها. يتم رسم خط رأسى متقطع من الكائن إلى أسفل المخطط يسمى «خط الحياة للكائن Object's Life Line". نرسم مستطيلاً رأسـياً ضيقاً حول خط الحياة للكائن يسمى «مستطيل التنشيط Activation"، وهو يمثـل تنفيـذ عملية يقوم بها الكائن. يمثل طول مسـتطيل التنشـيط الفترة الزمنية المطلوبة لتنفيذ العملية. يتم تمثيل الرسـالة من كائن إلى آخر بخط له سهم يبدأ من خط الحياة للكائن المرسل إلى خط الحياة للكائن المستقبل للرسالة.

يتم تمثيل الوقت فى المخطط فى الاتجاه الرأسى ويبدأ من أعلى متجهاً إلى أسفل. يتم تمثيل الرسـائل حسب الأسبقية، الرسـائل الموجودة أعلى المخطط تبدأ قبل التى تليها لأسـفل. وعلى هذا، فـإن مخطط التتابع يعتبر ثنائى الأبعـاد. تمثل الكائنات الاتجاه الأول (الأفقى) من اليسـار إلى اليمين، ويمثل الوقت الاتجاه الثانى (الرأسى) ويتجه من أعلى إلى أسفل.

مثال لمخطط التتابع:

يوضح شـكل (٢-٣٢) مثالاً لاستخدام مخطط التتابع، حيث يصف سيناريو تنفيذ إحدى حالات الاستخدام فى نظام طلبات العملاء الخاصة بحساب تكلفة أحد طلبات العمـلاء. يقوم الكائن an Order باسـتدعاء الكائن an Order Line لمعرفة السـلع المطلوبة وكميتها من خلال الرسـالتين getQuantity ، getProduct. وبعد ذلك يقوم الكائن an Order باسـتدعاء الكائن aProduct لمعرفة سـعر كل سـلعة موجودة فى الكائـن an Order Line من خلال الرسـالة getPricingDetails، ثم يقوم بحسـاب إجمالى السـعر لكل سلعة من خلال استدعاء الطريقة calculateBasePrice الموجودة

داخل الكائن نفسه an Order. وبعد ذلك يقوم الكائن an Order بالحصول على نسبة الخصـــم Discount من الكائن aCustomer من خلال الرســالة getDiscountInfo ليقـــوم بعد ذلك بحســاب التكلفة النهائية لطلب العميل. نلاحــظ فى هذا المثال أنه لم يتم تمثيل الرسائل المسترجعة لجميع الرسائل المرسلة فى المخطط، ولكن تم تمثيلها فـــى حالة واحدة فقــط، وذلك لتجنب كثرة الخطوط وتبقـــى الفكرة منطقية وهى أن جميع الرسائل من النوع Call تحتاج إلى رسائل مسترجعة لها.



شكل (٢-٣٢) مثال لمخطط التتابع

٢-٢-٧ مخطط الحالة State Diagram:

تمــر كائنات النظـــام بمجموعة من التغيرات فى أثناء تفاعلها مع المســتخدمين أو أنظمـــة أخـــرى. يمكن لأى كائــن أن يتفاعل مع كائنات أخرى فـــى النظام عن طريق

الرســائل المتبادلــة بينهم، مما يؤدى إلى نوع من التغيير فــى حالة الكائنات. وأحياناً يحــدث تغيير فى حالة الكائن مع مرور الوقت وبــدون تأثير من كائن آخر فيه. لابد من إيجاد وســيلة مناســبة لنمذجة تلك التغيرات التى تطرأ على الكائنات. تستخدم لغة UML مخطط الحالة باعتباره أســلوباً فعالاً لوصف السلوك الديناميكى لكائنات النظــام وذلك بتمثيل التغيرات التى تحدث لكائنات النظام مع الوقت أو نتيجة أحداث معينــة. فبعض الكائنات تتغير حالتها مع الوقت مثل أى عملية زمنية (مرتبطة بوقت محــدد). وبعض الكائنات تتغير حالتها نتيجة حدث معين مثل بدء تشــغيل أو إيقاف عملية أو إجراء معين.

يتكــون مخطــط الحالة مــن مجموعة مــن الحــالات States والانتقــالات بينها Transitions. وهو يصف رد فعل كائن معين للأحداث التى تواجهه. وبأسلوب آخر، فإن مخطط الحالة هو نموذج يصف جميع أشكال (حالات) كائن لصنف معين. فعند تعرض الكائن لحدث معين Event، فإنه يستجيب للحدث بتنفيذ إجراء يؤدى إلى تغيير حالته.

يعتبر مخطط الحالة هو منظور محلى لكائن يفصله عن باقى النظام، أى أنه يدرس ســلوك الكائن منعزلاً عما حوله. ولهذا فهو مناسب لدراسة وفهم سلوك كائن معين، ولكنه غير مناسب لدراسة سلوك النظام بأكمله.

الحدث Event:

الحــدث هو أى فعل يؤدى إلــى تبعات مؤثرة فى كائن آخــر. يمكن أن يصاحب الحدث مجموعة من المعاملات Parameters التى تحدد شكل التأثير. يتمثل التأثير فى شكل حدوث تغيرات على المتأثر. يمكن تصنيف الأحداث إلى أنواع عديدة منها: أحــداث النــداء Call Events، وأحداث التغيير Change Events، وأحداث الإشــارة Signal Events، وأحداث الوقت Time Events.

الحالة State:

يحتوى مخطط الحالة على مجموعة من الحالات لكائن واحد. يمكننا وصف حالة الكائن من خلال ثلاث خصائص هى:

- مجموعة من القيم للكائن.
- فترة زمنية ينتظر خلالها الكائن وقوع حدث معين.
- فترة زمنية يقوم خلالها الكائن بنشاط معين.

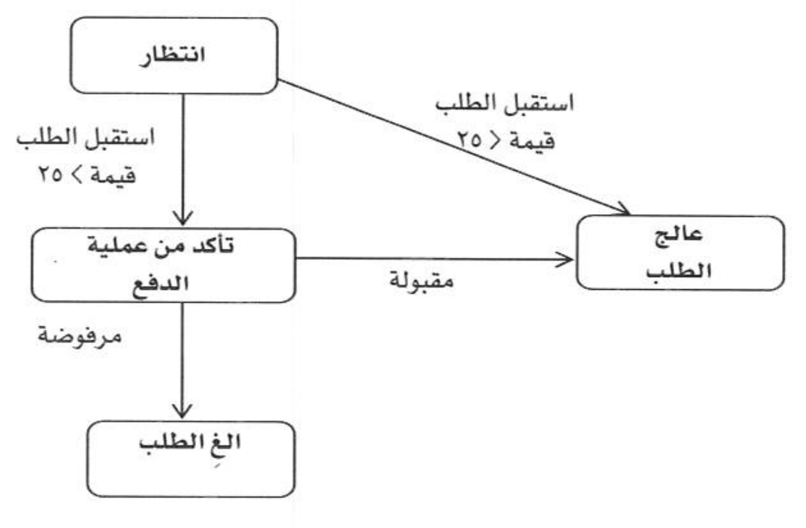
يمكــن أن تأخذ الحالة اســماً يصف تأثير الحدث فــى الكائن. يتم ربط الحالات بعضها ببعض عن طريق خطوط الانتقالات Transitions.

الانتقال Transition:

يحدث الانتقال دائماً من حالة إلى أخرى نتيجة للحدث الذى يؤثر فى الكائن ويغير من حالته. يمكننا وصف عملية الانتقال من خلال عدة عوامل، هى:

- زناد الحدث Event Trigger.
- شرط حماية Guard Condition.
- تأثير الحدث Event Effect.
- الحالة الهدف Target State.

يوضح شكل (٢-٣٣) مثالاً لبعض الانتقالات فى مخطط الحالة.



شكل (٢-٣٣) الانتقالات فى مخطط الحالة

مثال لمخطط الحالة:

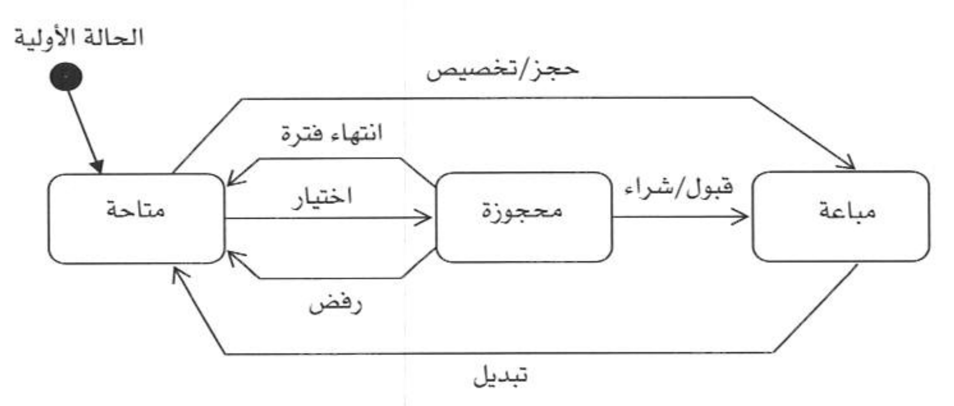
يوضح شكل (٢-٣٤) مثالاً لاستخدام مخطط الحالة لوصف الحالات التى يمر بها الكائن «تذكرة Ticket" فى نظام حجز وبيع التذاكر للعملاء لحضور عروض ترفيهية. تنحصر الحالات التى يمكن أن يأخذها الكائن فيما يلى:

- متاحة Available.
- محجوزة Locked.
- مباعة Sold.

يتعــرض الكائن «تذكرة Ticket" إلى مجموعة مــن الأحداث التى تؤدى إلى تغيير حالته مثل:

- حدث وقتى Time Out.
- حدث اختيار Select event.
- حدث رفض Reject event.
- حدث تغيير Change event.
- حدث موافقة Accept event.

فيبــدأ الكائــن «تذكــرة Ticket" دائماً بالحالــة «متاحة» ثم ينتقــل إلى الحالة «محجــوزة» فــى حالة إتمام عملية الحجــز، وينتقل إلى الحالة «مباعة» عند شــراء التذكرة فعلياً. ومن الممكن أن تنتقل حالة الكائن «تذكرة Ticket" من الحالة «مباعة» إلى الحالة «متاحة» عند اســترجاع أو تبديل التذكرة. وينتقل من الحالة «محجوزة» إلى الحالة «متاحة» عند انتهاء فترة الحجز المســموحة بدون شــراء التذكرة أو عند رفض الشراء.



شكل (٢-٣٤) مخطط الحالة للكائن Ticket

٢-٢-٨ مخطط النشاط Activity Diagram:

يعتبر مخطط النشاط هو أحد أساليب وصف منطق الإجراءات (العمليات) Process Logic أو لوصف تدفق العمل Work Flow. وهو يقوم بدور مشابه لما يقوم به مخطط التدفق Flow Chart مع اختلاف رئيسى بينهما وهو أن مخطط النشاط يسمح بالعمليات أو الوظائف المتوازية. ويتشابه مخطط النشاط مع مخطط التدفق فى استخدام نقاط القرار Decision Points والتفرعات Branches فى وصف خطوات العمليات. وجدير بالذكر هنا أن الخطوة Step فى مخطط التدفق تقابل النشاط Activity فى مخطط النشاط. يتم استخدام هذا المخطط بشكل كبير فى أثناء عملية التحليل لوصف منطق العمليات أو حالات استخدام النظام.

تستخدم لغة UML مخطط النشاط لوصف خطوات تنفيذ عملية أو سيناريو حالة استخدام داخل النظام فى شكل مجموعة من الأنشطة. ويعتبر أسلوب مخطط النشاط بديلاً لأسلوب الوصف النصى لسيناريو محتوى حالة الاستخدام. وهو يتميز بالوضوح وسهولة الفهم خصوصاً فى حالات احتواء السيناريو على مجموعة من التفريعات أو التكرارات التى يتم تمثيلها فى مخطط النشاط بشكل سهل ومفهوم مقارنة بالشكل النصى. تأخذ المسارات الموجودة بين مجموعة الأنشطة العديد من الأشكال منها ما يلى:

- التوالى Sequence.
- التفريع Branching.
- التوازى Concurrence.

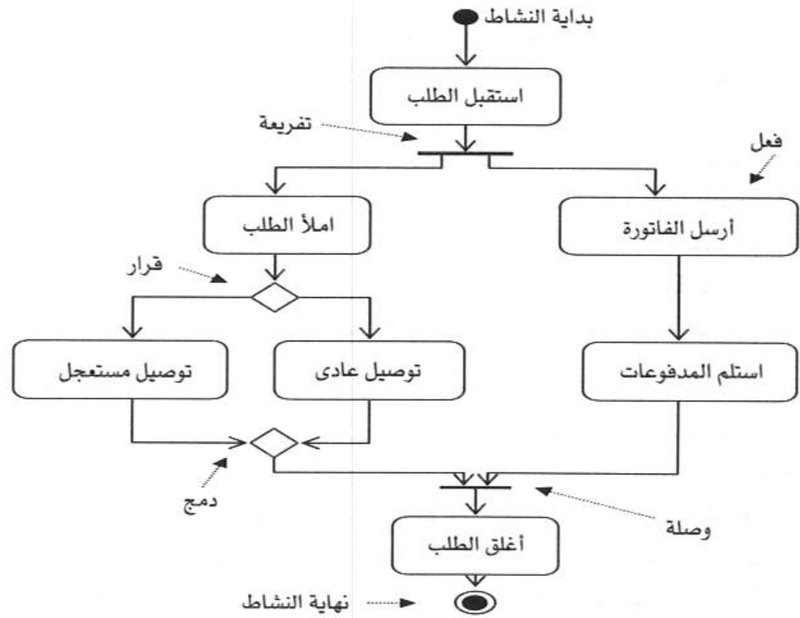
مثال لمخطط النشاط:

يوضح شكل (٢-٣٥) مثالاً لاستخدام مخطط النشاط فى معالجة طلبات العملاء.

نلاحظ فى المثال أنه بعد استقبال الطلب يتم اتخاذ مسارين متوازيين هما:

- تعبئة الطلب Fill Order، توصيل الطلب Order Delivery.
- إرسال فاتورة Send Invoice، استلام مدفوعات Receive Payment.

لأن كلاً من المسارين لا يعتمد أحدهما على الآخر، فمن الممكن تنفيذهما فى وقت واحد. وبعد ذلك يتم التقاء (دمج) المسارين لتكملة الإجراءات بالنشاط Close Order. ففى المسار الأول يتم تحديد نوع الطلب بعد عملية التعبئة، فإذا كان الطلب سريعاً Rush Order، يتم تنفيذ خطوة التسليم السريع، وإذا كان الطلب عادياً، يتم تنفيذ خطوة التسليم العادى. بالطبع يختلف هذا السيناريو من نظام لآخر، ويسمح مخطط النشاط بتمثيل أى أسلوب وكذلك ترتيب النشاطات ومساراتها المختلفة.



شكل (٢-٣٥) مثال لاستخدام مخطط النشاط.

٢-٢-٩ مخطط المكونات Components Diagram:

يستخدم مخطط المكونات لوصف مكونات النظام والعلاقات بينها، ولتوضيح ذلك نبدأ بتعريف مكونات النظام.

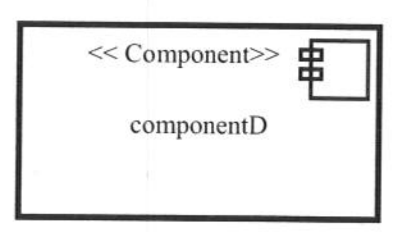
المكون Component:

المكون هو تمثيل لأحد المنتجات البرمجية للنظام مثل الملفات التنفيذية Executable Files، وواجهات المستخدم User Interface، وملفات التوثيق Documentation Files، وملفـــات البيانـــات Data Files وغيرهـــا . يتم تمثيل المكون فى لغة UML فى شـــكل مســـتطيل يحتوى على علامة توضيحية تميزه وهى <<component>> أو علامة تدل علـــى نوع المكون مثـــل <<Executable>> كما فى شـــكل (٢-٣٦). ويمكن إضافة رمز المكون المســـتخدم فى الإصدار الأول للغـــة النمذجة الموحدة UML1 كما هو مبين فى شكل (٢-٣٧) (رسمى، ٢٠٠٥).

<< Executable >>
componentB

<< Component>>
componentC

شكل (٢-٣٦) تمثيل المكون فى لغة UML2

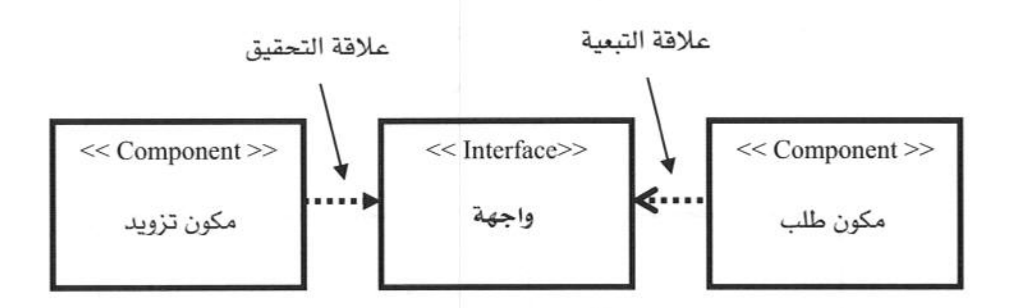


شكل (٢-٣٧) تمثيل آخر للمكون فى لغة UML2

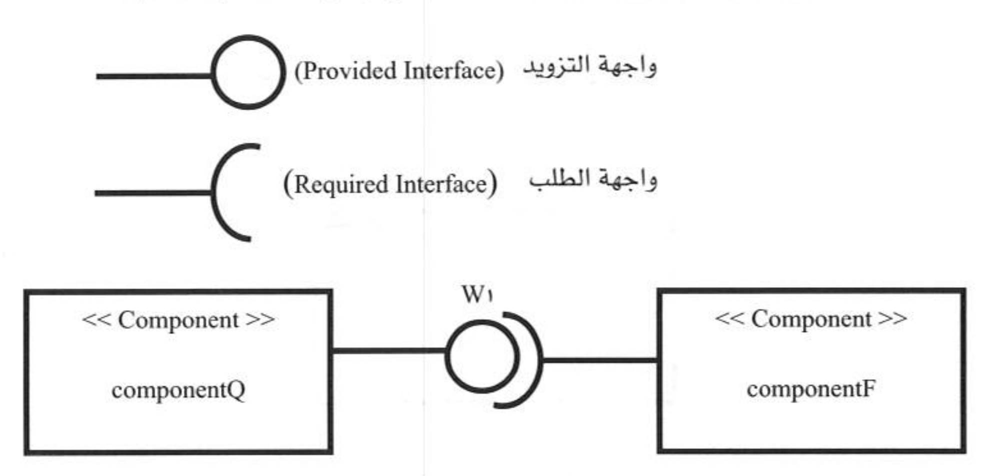
الواجهات والمكونات Components and Interfaces:

الواجهة Interface عبارة عن مجموعة من العمليات التى يمكن من خلالها الوصول أو الاتصـــال بأطراف أخرى أو مكونات أخرى داخل النظام. يطلق على واجهة المكون الذى يوفر الخدمات واجهة التزويد Provide Interface. وتســـمى العلاقة بين المكون

والواجهة «علاقة التحقيق Realization"، أى أن الواجهة هى تحقيق للمكون. ويطلق على واجهة المكون الذى يطلب الخدمات «واجهة الطلب Require Interface"، وتسمى العلاقة بين الواجهة والمكون «علاقة التبعية Dependency". يتم تمثيل هذه العلاقات بطريقتين موضحتين فى شكل (٢-٣٨) وشكل (٢-٣٩).



شكل (٢-٣٨) تمثيل علاقات التحقيق والتبعية باستخدام الأسهم



شكل (٢-٣٩) تمثيل علاقات التحقيق والتبعية باستخدام المكبس والكرة

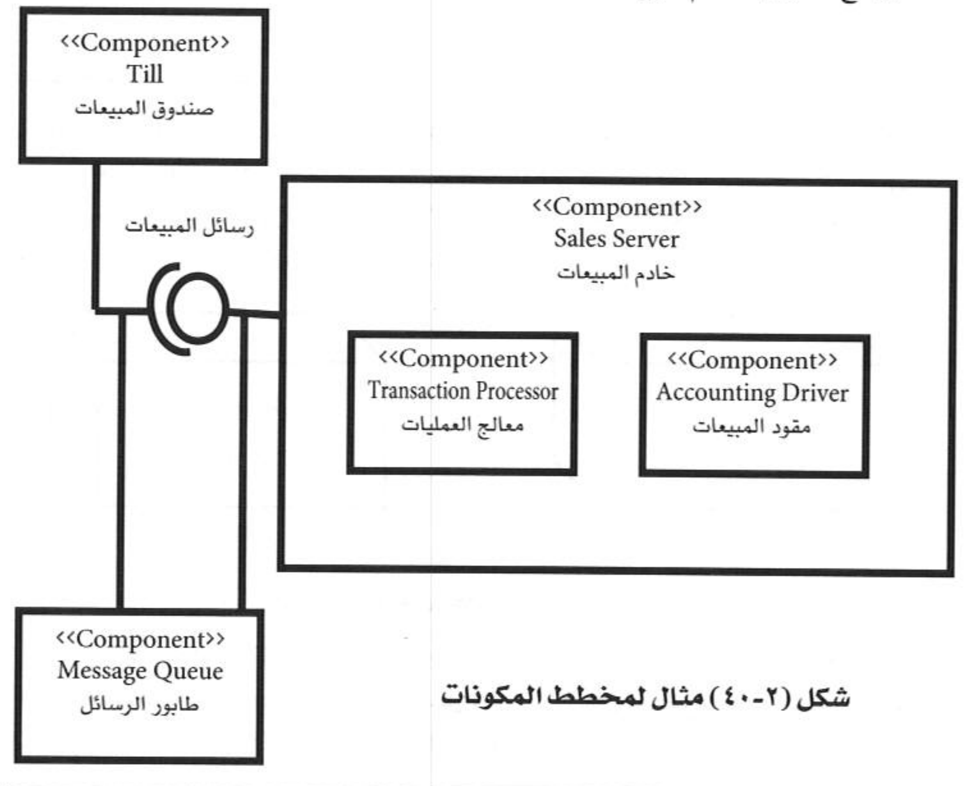
مثال لمخطط المكونات:

تتطلب عملية تصميم مخطط المكونات أن نقوم أولاً بتحديد مكونات النظام، وهى كما ذكرنا ســابقاً عبارة عن منتجات برمجية ينتجها أو يســتخدمها النظام. بعد ذلك

نقــوم بتحديد واجهات المكونات Interfaces، ثم نقــوم بوصف العلاقات المختلفة بين المكونات والواجهات (التحقيق والتبعية). يوضح شكل (٢-٤٠) مثالاً لمخطط المكونات لأحد أنظمة المبيعات وهو يتكون من المكونات التالية:

- خادم المبيعات Sales server.
- صندوق المبيعات Till.
- طابور الرسائل Message Queue.

نلاحــظ فى المثال علاقة الربط بين مكون «صنــدوق المبيعات Till" ومكون «خادم المبيعات Sales Server" من خلال واجهة يطلق عليها اســم «رسائل المبيعات Sales Message". ونلاحــظ أن المكون «طابور الرســائل Message Queue" يقوم بعمليتين فى المخطط، فهو يقوم بتزويد واجهة «رسائل المبيعات Sales Message" للتعامل مع مكون «صندوق المبيعات Till"، ويقوم أيضاً بطلب الخدمة من واجهة «رسائل المبيعات Sales Message" للتعامل مع المكون «خادم المبيعات Sales Server".



شكل (٢-٤٠) مثال لمخطط المكونات

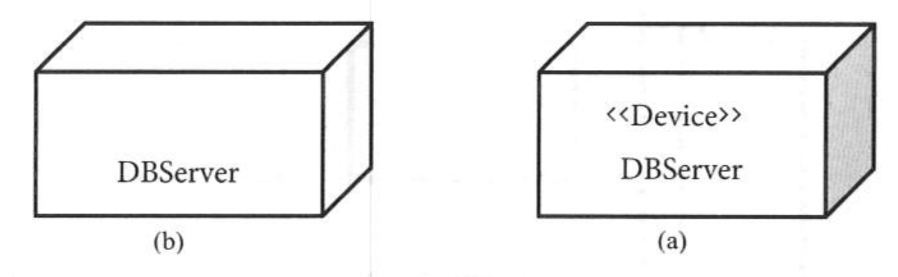
٢-٢-١٠ مخطط النشر:

يقوم مخطط النشـــر بتمثيل عملية نشر المنتجات البرمجية فى أثناء وقت التنفيذ Run-Time من خلال مجموعة من العقد Nodes. تمثل كل عقدة أحد الموارد المادية للعتاد Hardware المستخدم فى تنفيذ المنتجات البرمجية للنظام مثل المعالج Processor، والذاكــرة Memory، والوحدات الطرفية Peripheral Units. يقوم مخطط النشـــر أيضاً بتوضيح العلاقات بين المكونـــات البرمجية Software Components والمكونات المادية Hardware Components ويوضح كذلك كيفية الاتصال بين العقد.

يســـتخدم مخطط النشـــر عند الحاجة لمعرفة مكان نشـــر المكونات البرمجية فى النظام. ويســـتخدم أيضاً فى بناء الأنظمة الموزعة Distributed systems؛ لأنه يقوم بتوضيح العلاقات بين المكونات البرمجية الموزعة ومكان نشـــرها على المكونات المادية، مثل تطبيقات نقاط البيع Point-of-Sales، التى تنفذ من خلال شبكة حاسب موسعة تتضمن مجموعة من العملاء Clients المنتشـــرين فى أماكـــن متفرقة ومتصلة بخادم مركزى أو أكثر من خادم Servers.

تصميم مخطط النشر:

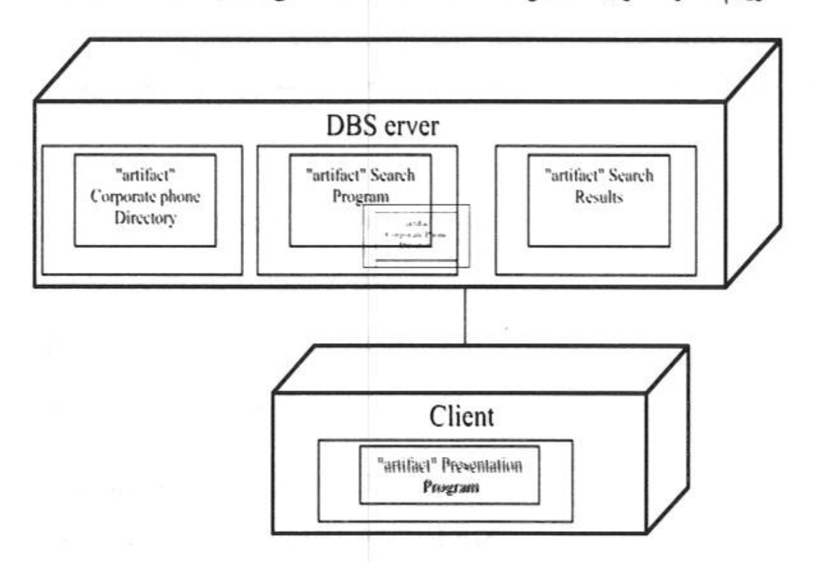
يتكون مخطط النشر من مجموعة من العقد التى تمثل المكونات المادية للنظام. يتم تمثيل العقدة Node فى لغة UML باستخدام رمز المكعب وتأخذ اسماً محدداً ويمكن استخدام الكلمة المحجوزة <<Device>>، كما هو موضح بالشكل (٢-٤١) لتمثيل مكون خادم قواعد البيانات DBServer (رسمى، ٢٠٠٥).



شكل (٢-٤١) تمثيل العقدة Node **فى لغة** UML

يتم تمثيل عملية النشـــر للمنتجات البرمجية من خلال مجموعة من العقد وكيفية الاتصـــال بينها، كما هـــو مبين بالشـــكل (٢-٤٢) الذى يوضح عملية نشـــر المنتجات البرمجية لنظام «دليل الهاتف Telephone Directory" كما يلى:

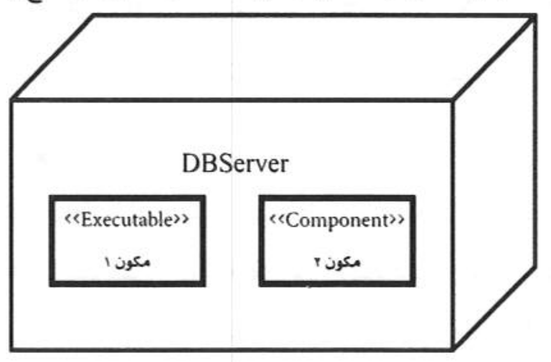
يتم نشـر المكونات "Corporate Phone Directory" "Search Result"، "Search Program" على العقدة، "DBServer" ويتم نشر المكون "Presentation Program" على العقدة "Client".



شكل (٢-٤٢) تمثيل عملية نشر المنتجات البرمجية

دمج مخطط المكونات مع مخطط النشر:

يمكننا دمج مخطط المكونات فى مخطط النشر للحصول على مخطط يصف مجموعة المكونات البرمجية التى تقوم بتنفيذها العقد الممثلة للمكونات المادية. وبمعنى آخر فإن العقدة تمثل الموقع الذى يعمل فيه المكون، ولذلك فإنه يتم تثبيت المكون على العقدة، وتسمى تلك العملية «عملية نشر المكونات»، كما هو موضح بشكل (٢-٤٣).



شكل (٢-٤٣) دمج مخطط المكونات فى مخطط النشر

مثال لمخطط النشر:

سوف نقوم الآن بتصميم مخطط النشر لنظام إدارة الطلبة Students Administration. ولتمثيل ذلك سوف نبدأ بتحديد العقد المادية للنظام، وكذلك المكونات البرمجية، ثم نقوم بتمثيل العلاقات بينها من خلال مخطط النشر كما يلي:

١- العقد Nodes:

- خادم التطبيقات Applications Server.
- خادم قواعد البيانات DB Server.
- جهاز حاسب رئيسى Main Frame.
- خادم الويب Web Sever.

٢- المكونات البرمجية Components:

- إدارة الطلاب Students Administration.
- قاعدة البيانات الجامعية University DB.
- إدارة المقررات Courses Management.

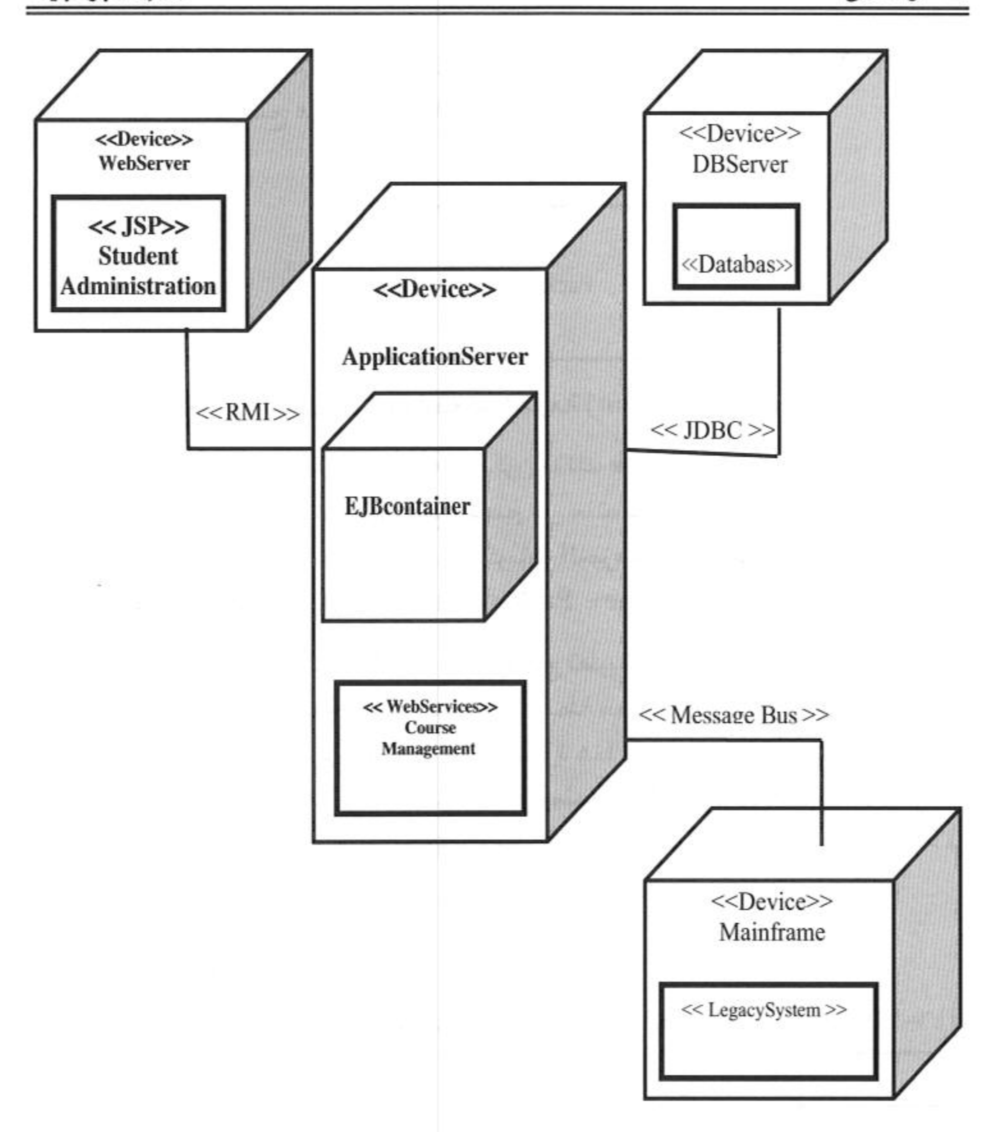
٣- الارتباطات بين العقد والمكونات:

- مكون إدارة الطالب مرتبط بعقدة خادم الويب.
- مكون قاعدة البيانات الجامعية مرتبط بعقدة خادم قواعد البيانات.
- مكون إدارة المقررات مرتبط بعقدة الحاسب الرئيسى.

٤- الاتصالات بين العقد:

- تتصل عقدة خادم التطبيقات بعقدة خادم الويب من خلال برتوكول لغة جافا RMI.
- تتصل عقدة خادم التطبيقات بعقدة قواعد البيانات من خلال برتوكول JDBC.
- تتصل عقدة خادم التطبيقات وعقدة الحاسب الرئيسى من خلال ناقل الرسائل Message Bus.

يوضح شكل (٢-٤٤) مخطط النشر لنظام إدارة الطلبة.



شكل (٢-٤٤) مخطط النشر لنظام إدارة الطلبة

٢-٢-١١ مخطط الحزم Packages Diagram:

يعتبر الهدف من تصميم الحزمة Package هو تجميع لبعض مكونات النظام، مثل الأصناف Classes أو حالات الاستخدام Use Cases وهى تكون غالباً مرتبطة وتؤدى

غرضـاً محـدداً. إذ يتم إحاطة هـذه العناصر المجمعة، فى أيقونـة مجلد Tabbed Folder، كما هو موضح فى الشكل (٢-٤٥).



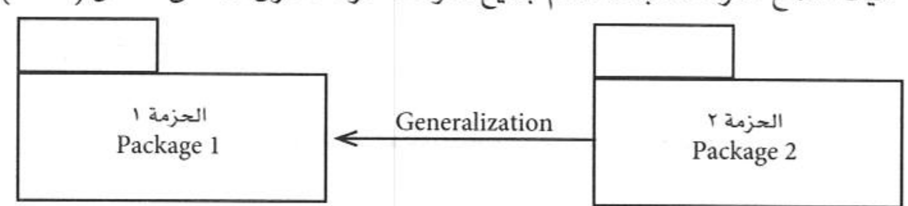
الشكل (٢-٤٥) أيقونة الحزمة (Package Icon)

وعند إعطاء اسـم للحزمة Package نكون بذلك قد أعطينا اسماً للمجموعة التى تم تجميعها، وليس شرطاً أن يتم وضع الاسم فى منتصف الأيقونة، يمكن كتابة الاسم فى العلامة المجدولة. ولاستخدام أحد مكونات الحزمة، يجب استخدام الرمز التالى (::)، كما فى الصيغة التالية: PackageName :: PackageElement

وتستخدم الحزمة لتنظيم مكونات أى برنامج تطبيقى (Application Program)، إذ إن الحزمة تمثل مجموعة من الأصناف المترابطة معاً بقوة، ولكنها ضعيفة الارتباط نسـبياً بالمجموعات الأخرى. ولا بد من الإشارة هنا إلى أنه يمكن أن ينتمى الصنف الواحـد إلـى حزمة واحدة فقط، لكن هذا لا يمنعه من الظهور فى حزم أخرى أو من الاتصـال مع أصنـاف فى حزم أخرى. يمكن أن ترتبط الحـزم معاً من خلال ثلاثة طرق هى:

١- التعميم (Generalizations):

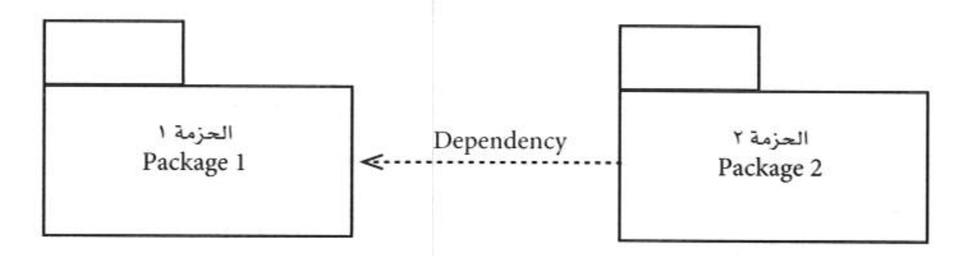
حيث تسمح لحزمة ما باستخدام جميع مكونات حزمة أخرى كما فى الشكل (٢-٤٦).



الشكل (٢-٤٦) علاقة التعميم بين الحزم

٢- التبعية (Dependency):

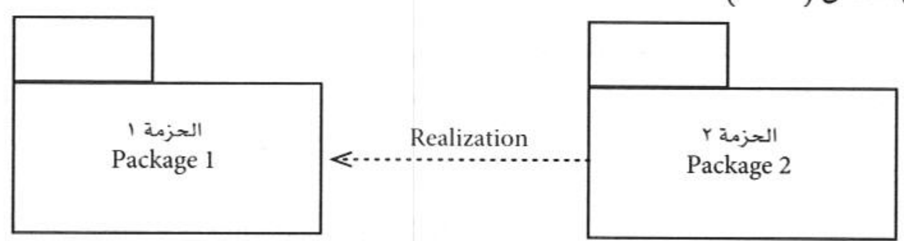
حيـــث تعتمـــد مكونات فى حزمة ما على مكونات فى حزمة أخرى، أى أن أى تغيير يحدث فى مكونات الحزمة يؤثر فى الحزمة المعتمدة عليها، كما فى الشكل (٢-٤٧).



الشكل (٢-٤٧) علاقة التبعية بين الحزم

٣- التحقيق (Realization):

حيث تقوم مكونات حزمة ما بتحقيق أو تطبيق مكونات موجودة بحزمة أخرى، كما فى الشكل (٢-٤٨).



الشكل (٢-٤٨) علاقة التحقيق بين الحزم

ما هو مخطط الحزم Packages Diagram؟

إن مفهوم مخطط الحزم غير موجود بذاته فى لغة UML. ولكن يتم استخدامه فى بناء مخطط الأصناف Classes Diagram أو مخطط حالات الاســتخدام Use Cases Diagram للأنظمة الكبيرة الحجم وذلــك لغرض تنظيمى فقط نظراً لاحتوائها على عدد كبير من المكونات. إذ يتم إنشــاء حزم تحتوى على أصناف أو حالات اســتخدام، وبعد ذلك نقوم بإنشاء مخطط أصناف تظهر فيه الحزم المخزن فيها الأصناف، ونسمى

المخطــط فى هذه الحالة مخطط حزم الأصناف Classes Package Diagram، وذلك لتســهيل عملية فهم مكونات النظام. وكذلك نقوم بإنشاء مخطط حالات الاستخدام، تظهر فيه الحزم المخزن فيها حالات الاســتخدام، ويســمى المخطط فى هذه الحالة، مخطط حزم حالات الاستخدام Use Cases Package Diagram.

متى نستخدم مخطط الحزم Packages Diagram؟

تعتبر الحزم من الأدوات الحيوية للمشــاريع الكبيرة، لذلك نستخدم الحزم عندما يكون مخطط الأصناف كبيراً جداً، ولا نســتطيع وصفــه فى مخطط واحد، وعندها نقــوم بتجميع مجموعة الأصنــاف المرتبطة معاً ووضعها فى حزمــة واحدة، وتجميع الأصناف التى تشــترك فى النوع نفسه من التوريث فى حزمة أخرى، وهكذا. وينصح الكثيــر من المصممين ومطورى النظم باســتخدام لغة UML، أن يتم بناء واســتخدام مخطط الحزم فقط عندما نشعر بأنه سيؤدى إلى وصف بسيط ومفهوم ومفيد لبنيان النظــام، وأيضاً عندما نريد أن نبقى المكونــات أو أجزاء النظام صغيرة قدر الإمكان، للمحافظة على بنيانه وسهولة متابعته (بالمقارنة بمخطط المكونات).

ويظهر ذلك واضحاً، عند استخدام الحزم فى مخطط حالات الاستخدام، إذ تكمن أهميتــه فــى إدارة حجم وتعقيد المكونات الوظيفية للنظام، كما أنه يشــكل آلية مهمة وأساسية للتعاون بين مطورى النظام.

مثال لمخطط الحزم Packages Diagram:

سنقوم الآن ببناء مخطط حزم الأصناف لجزء من نظام التسجيل الجامعى.

يتكون هذا المخطط من ثلاث حزم هى:

١- حزمة التقييمات (Marking Package):

تحتوى على جميع الأصناف المرتبطة التى لها علاقة بتسجيل درجات التقييمات للطلاب.

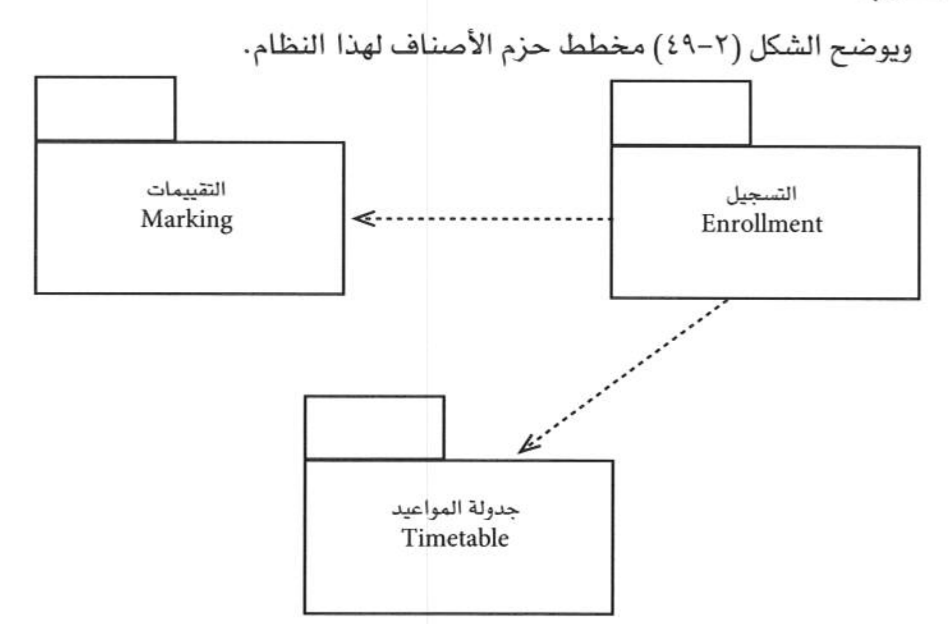
٢- حزمة التسجيل (Enrollment Package):

تحتوى على جميع الأصناف المرتبطة التى لها علاقة بعمليات التسجيل للطلاب.

٣- حزمة جدول المواعيد (Timetable Package):

تحتــوى على جميع الأصناف المرتبطة التى لها علاقة بجدولة مواعيد المحاضرات للطلاب.

ويوضح الشكل (٢-٤٩) مخطط حزم الأصناف لهذا النظام.



شكل (٢-٤٩) مخطط حزم الأصناف لجزء من نظام التسجيل الجامعى

نلاحظ من الشكل وجود علاقة التبعية فى المخطط، إذ تعتمد حزمة التسجيل على كل من الحزمتين: حزمة التقييمات وحزمة جدول المواعيد.



الفصل الثالث
أدوات هندسة البرمجيات
Software Engineering Tools

تحتاج عمليات هندسة البرمجيات إلى استخدام العديد من الأدوات فى أثناء مراحل التطوير المختلفة وأيضاً فى أثناء تنفيذ أنشطة إدارة المشروع. يؤدى الاستخدام الجيد لهذه الأدوات إلى تحسين أداء وإنتاجية الأفراد ومن ثم يعود بالنفع على فريق التطوير نفسه وعلى المشروع كله من خلال رفع كفاءة وجودة المنتجات البرمجية. وعلى هذا فإن استخدام أدوات هندسة البرمجيات يتطلب قدراً معيناً من المعرفة والمهارة. يتمثل الدعم الذى تقدمه هذه الأدوات لعمليات التطوير وإدارة المشروع فى استخدامها فى أثناء عمليات نمذجة النظام، وبناء وتطوير البرمجيات، ومهام إدارة المشروع. ولذلك فإنه يتم تصنيف أدوات هندسة البرمجيات إلى المجموعات التالية:

- أدوات إدارة المشاريع.
- أدوات نمذجة النظم.
- أدوات برمجة التطبيقات.

ومن المعروف أن هذه الأدوات يتم التحديث فيها بشكل مستمر لتواكب التطور فى أساليب ومنهجيات تطوير النظم وكذلك التقنية الحديثة. سوف نستعرض الآن بعض الأدوات المهمة التى تستخدم على نطاق واسع فى أثناء تطوير وإدارة مشاريع البرمجيات.

٣-١ أدوات إدارة مشاريع البرمجيات Software Project Management Tools:

تقوم عمليات إدارة مشاريع البرمجيات على استخدام مجموعة من الأدوات والأساليب والطرق المتعلقة بتخطيط ومتابعة إنجاز المهام والعمليات الخاصة بالمشروع. وسوف نتطرق فى الفصل القادم لمجموعة من الطرق والأساليب المستخدمة فى إدارة المشروع، أما الآن فسوف نركز على عرض لبعض الأدوات المستخدمة فى إدارة المشروع. ويجب الإشارة هنا إلى أن تلك الأدوات تستخدم لإدارة أى نوع من أنواع المشاريع سواء

كانت مشـــاريع برمجيات أو مشـــاريع أخرى. تركز إدارة المشروع على عمليات تنظيم وإدارة الموارد البشرية والمادية المطلوبة لإنجاز مهام المشروع فى فترة محددة ومن خلال ميزانيـــة محددة. وعلى هذا فإن الإدارة الجيدة للمشـــروع تتطلب التخطيط والمتابعة الجيدة للسيطرة على ثلاثة عناصر أساسية بالمشروع هى:

- الموارد Resources.
- الوقت Time.
- الميزانية Budget.

ويتطلب ذلك توافر مجموعة من الأدوات الفعالة التى تســاعد فى تنفيذ أنشـطة إدارة المشروع المختلفة (مثل جدولة المهام، توزيع الموارد، تقدير التكلفة، مراقبة عمليات التنفيذ، غيرها) بشكل جيد. لقد ظهرت فى الآونة الأخيرة بعض الأدوات الحديثة التى تعمل من خلال بيئة متكاملة لإدارة المشـــروع، حيث تتضمن معظم الأدوات الضرورية الخاصة بتخطيط وتنظيم ومتابعة المشـــروع. تتســـم بعض هذه الأدوات بقدرتها على العمـــل من خلال شـــبكة الإنترنت التى تمكـــن مديرى المشـــروعات الكبيرة من تنفيذ مهامهم بشكل تفاعلى وفى الوقت الفعلى Real Time. فيما يلى عرض لبعض أدوات إدارة المشاريع.

عنوان الموقع على الإنترنت	اسم الأداة
http://www.microsoft.com/office/project	MS Project (Microsoft, 2006)
http://www.managepro.net	Manage Pro (ManagePro, 2006)
http://www.eroom.net/eRoomNet	eRoom (eRoom, 2006)
http://www.eproject.com	eProject (eProject, 2006)
http://www.primavera.com	Primavera Enterprise (Primavera, 2006)
http://www.thesmallworlds.com	Small Worlds (SmallWorlds, 2006)
http://www.palisade-europe.com	@Risk (Palisade, 2006)
http://www.iceincusa.com//products_tools	Risk Radar (ICE, 2006)

٣-١-١ جدولة المشروع Project Scheduling:

هناك العديد من الأدوات التى تستخدم فى جدولة أنشطة المشروع ومتابعتها من خلال إعداد مجموعة من المخططات التى تصف الأنشطة ومواعيد بدء كل نشاط والفترة التى يستغرقها النشاط، وكذلك تخصيص الموارد لكل نشاط، وغيرها. يتطلب ذلك تجهيز المعلومات الأساسية التالية:

- تحديد الأنشطة أو المهام Activities or Tasks.
- تحديد الفترة الزمنية لكل مهمة أو نشاط.
- ترتيب تنفيذ المهام.
- تخصيص الموارد لكل نشاط:

بعد ذلك يتم جدولة مهام المشروع باستخدام المعلومات المجهزة فى شكل مخطط يدعم عملية متابعة إنجاز المهام خلال مراحل التطوير المختلفة. سوف نستعرض الآن طريقتين من أكثر الطرق استخداماً فى أثناء جدولة ومتابعة مشاريع البرمجيات وهما مخطط جانت GANTT chart، ومخطط بيرت PERT chart. تساعد هذه الأدوات مدير المشروع فى عمليات ضبط وتعديل جدول المشروع حسب التغيرات التى تطرأ على أنشطة وموارد المشروع فى أثناء عمليات التنفيذ.

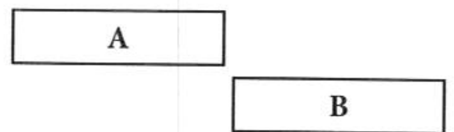
٣-١-١-١ مخطط جانت GANTT Chart:

يعتبر مخطط جانت من أكثر الطرق شيوعاً أثناء عمليات جدولة ومتابعة مهام المشروع. يتم فيها تحديد بداية ونهاية كل مهمة وتمثيلها فى المخطط حسب ترتيبها الزمنى داخل المشروع فى شكل مستطيل فارغ. وبعد ذلك يتم استخدام المخطط فى أثناء عمليات متابعة تنفيذ مهام المشروع وذلك من خلال تظليل الأجزاء المنفذة فعلياً فى كل مهمة ومقارنتها بما هو مخطط مسبقاً فى الجدول. ويساعد ذلك مدير المشروع فى اتخاذ الأفعال التصحيحية المناسبة لتدارك أى تأخير أو عقبات تحدث لأى مهمة وعدم تراكم تأثيرها فى الأنشطة التالية لها.

هناك ثلاثة أشكال لترتيب أنشطة المشروع هى:

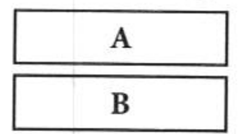
١- توالى Sequence:

حيث يعتمد نشــاط معين اعتماداً كلياً على نشاط آخر يكون سابقاً له فى الإنجاز كضرورة لبدء تنفيذه.



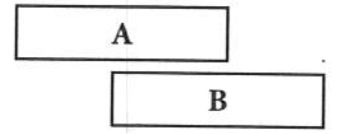
٢- توازى Parallel:

فيها يمكن أن يتم تنفيذ نشــاطين أو أكثر فى الوقت نفسه، أى أنه لا يوجد اعتماد لأحد الأنشطة على إنجاز نشاط آخر.



٣- تداخل Overlap:

فيها يمكن أن يبدأ نشــاط معين بعد بداية نشــاط آخر بفترة ولكن لا يبدأ معه فى الوقت نفسه.



أى أنه لا ينتظر لاستكمال النشاط الآخر حتى يبدأ هو فى التنفيذ، ولكن يمكنهما الاشتراك فى فترة زمنية معينة.

تتسم خريطة جانت بالمميزات التالية:

- سهولة التطبيق.
- إظهار الفترة الزمنية لكل نشاط مما يتيح عملية المراقبة أثناء التنفيذ.
- إظهار مدى التداخل الزمنى بين الأنشطة بشكل مرئى وسهل.

مثال لاستخدام مخطط جانت:

سـوف نقترح هنا مشروع تطوير نظام معلومات لإحدى المؤسسات التجارية. يقوم المشــروع بتطبيق أنشــطة دورة حياة تطوير النظم بدءاً من تحديـد المتطلبات وانتهاءً بتثبيت النظام. الفترة الزمنية المخصصة للمشــروع هى (٢٤) أســبوعاً. والآن سوف نستعرض خطوات جدولة أنشطة المشروع.

١- تحديد أنشطة المشروع:

لا تختلف أنشــطة نظم المعلومات كثيراً من مشــروع لآخر؛ لأنها تطبق الأنشــطة المطلوبة لتحقيق دورة حياة تطوير نظم المعلومات وهى:

- تحديد الاحتياجات Requirement Determination.
- تحليل العمليات Processes Analysis.
- تحليل البيانات Data Analysis.
- تصميم العمليات Processes Design.
- تصميم قواعد البيانات Data base Design.
- تصميم الشاشات Screen design.
- تصميم التقارير Report Design.
- برمجة التطبيقات Applications Programming.
- اختبار البرامج Testing Programs.
- تثبيت النظام System Installation.

٢- تقدير زمن كل نشاط:

هناك ثلاثة تقديرات تستخدم لتحديد الزمن المتوقع لإتمام إنجاز أى مهمة هى:

- تقديـر الوقت المتفائـل Optimistic Time (O)، وهو أقل فترة ممكنة لإتمام إنجاز النشاط.
- تقدير الوقت المتشائم Pessimistic Time (P)، وهو أكبر فترة ممكنة لإتمام إنجاز النشاط.

- تقديـــر الوقت الواقعى Realistic Time (R)، وهو يمثل أفضل تقدير من وجهة نظر مدير المشروع للفترة الزمنية المعقولة لإتمام إنجاز النشاط.

تستخدم تلك التقديرات لحساب الفترة الزمنية المتوقعة لإتمام إنجاز النشاط كما يلى:

$$ET = (O + 4R + P) / 6$$

نلاحــظ من تلك المعادلة أن الوقت الواقعى أخذ وزناً يكافئ أربعة أضعاف وزن كل من الوقت المتفائل والوقت المتشائم نظراً لقربه من الوقت الفعلى للنشاط. يمثل شكل (٣-١) تقديرات الوقت المتوقع لأنشطة المثال الحالى بالأسبوع.

وصف النشاط	الوقت المتفائل O	الوقت المتشائم P	الوقت الواقعى R	الوقت المتوقع ET
تحديد الاحتياجات	١	٣	٢	٢
تحليل العمليات	١	٥	٣	٣
تحليل البيانات	١	٩	٢	٣
تصميم العمليات	٤	١٤	٦	٧
تصميم قواعد البيانات	٢	١٨	٤	٦
تصميم الشاشات	١	١	١	١
تصميم التقارير	٢	٨	٥	٥
برمجة التطبيقات	٣	٥	٤	٤
اختبار البرامج	٥	١٥	٧	٨
تثبيت النظام	١	٣	٢	٢

شكل (٣-١) تقديرات الوقت المتوقع لأنشطة أحد المشاريع

٣- ترتيب أنشطة المشروع:

وهنا يتم تحديد الترتيب الزمنى لأنشــطة المشــروع وعلاقة كل نشاط بالسابق له والتالى له. يوضح شكل (٣-٢) الترتيب الزمنى لأنشطة المثال الحالى.

نلاحظ أن الأنشــطة رقم (٣)، (٤)، (٥) تعمل بالتــوازى، أى أنها تعمل فى الوقت نفسه؛ لأن النشاط السابق لها واحد وهو النشاط رقم (٢). ونلاحظ أيضاً أن النشاط رقم (٦) يعتمد على كل من النشاطين (٣)، (٤) أى أنه لا يبدأ إلا بعد الانتهاء من تنفيذ

كلا النشاطين، ولذلك فهو يبدأ فى الخطة بعد انتهاء النشاط (٤) حيث إنه يحتاج إلى وقت أكبر من النشاط (٣)، وكذلك الحال فى الأنشطة رقم (٧)، (٨)، (١٠).

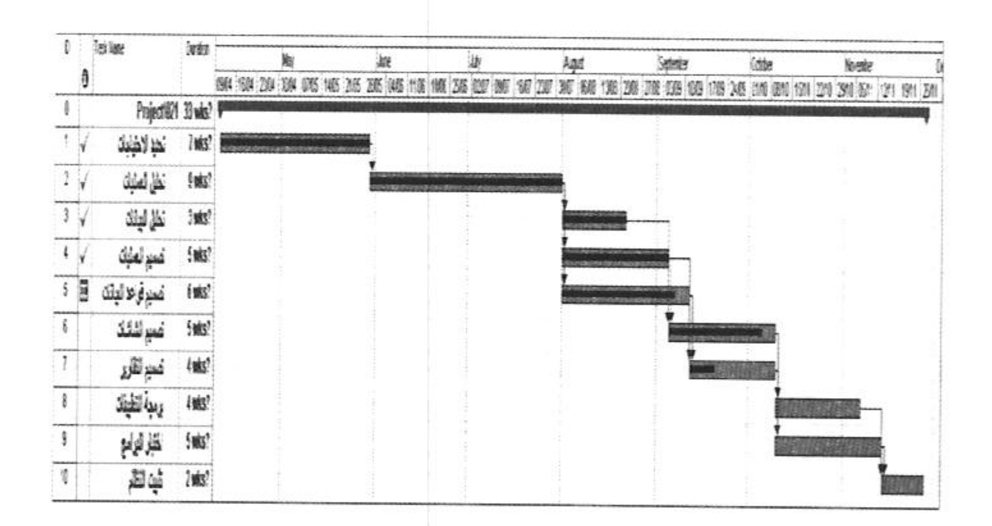
النشاط الحالى	وصف النشاط	النشاط السابق
١	تحديد الاحتياجات	-
٢	تحليل العمليات	١
٣	تحليل البيانات	٢
٤	تصميم العمليات	٢
٥	تصميم قواعد البيانات	٢
٦	تصميم الشاشات	٤ و ٣
٧	تصميم التقارير	٥ و ٤
٨	برمجة التطبيقات	٧ و ٦
٩	اختبار البرامج	٧
١٠	تثبيت النظام	٩ و ٨

شكل (٣-٢) الترتيب الزمنى لأنشطة أحد المشاريع

٤- رسم خريطة جانت للمشروع:

يســتخدم كل من التقدير الزمنى لأنشــطة المشــروع والترتيب الزمنى لها لرسم خريطة جانت لأنشطة المشروع. يوضح شكل (٣-٣) خريطة جانت للمشروع الموضح بالمثال الحالى باستخدام برنامج Microsoft Project (السماك، ٢٠٠٤).

Company	معهد الإدارة العامة
Current Date	٢٠٠٦/١٠/٨
Title	مثال على إدارة مشاريع البرمجيات
Project Start	٢٠٠٦/٠٤/١١ ٠٨:٠٠:٠٠ ص
Project Finish	٢٠٠٦/١١/٢٧ ٠٥:٠٠:٠٠م
Complete ٪	٪٧٠



شكل (٣-٣) استخدام برنامج Microsoft Project **لإعداد مخطط جانت**

٣-١-١-٢ مخطط بيرت PERT Chart:

هو أســلوب آخر لتمثيل جدولة نشاطات المشروع بوصف الترتيب المنطقى والزمنى لأنشطة المشروع وعلاقة كل نشاط بالنشاطات الأخرى من ناحية اعتماده عليه كلياً أو جزئياً أو عدم الاعتماد عليه إطلاقاً عند بداية تنفيذ النشــاط. يتكون مخطط بيرت من مكونين أساســين هما مجموعة من العقد Nodes تمثل الأنشــطة، ومجموعة من الأســهم Arrows تمثل ترتيب الأنشطة. تحتوى كل عقدة فى المخطط على مجموعة من المعلومات عن النشــاط الذى تمثله. يوضح شــكل (٣-٤) الشكل العام للعقدة فى مخطط بيرت.

يمكننا حساب قيم الأزمنة الأربعة لكل نشاط كما يلى:

زمن البداية المبكرة للنشاط = زمن البداية المبكرة للنشاط السابق + زمن النشاط السابق.

زمن النهاية المبكرة للنشاط = زمن البداية المبكرة للنشاط + زمن النشاط.

زمن البداية المتأخرة للنشاط = زمن البداية المتأخرة للنشاط التالى - زمن النشاط.

زمن النهاية المتأخرة للنشاط = زمن البداية المتأخرة للنشاط + زمن النشاط.

زمن النهاية المبكرة	زمن النشاط	زمن البداية المبكرة
	اسم النشاط	
زمن النهاية المتأخرة	الوقت الفائض	زمن البداية المتأخرة

شكل (٣-٤) الشكل العام للعقدة فى مخطط بيرت

ويمكن حساب زمن البداية المبكرة للنشاط Early Start Time (T) بتجميع الوقت المقدر لكل نشاط من بداية الشبكة من اليسار إلى اليمين حتى نصل للنشاط نفسه. وإذا كان هناك نشاط يعتمد على أكثر من نشاط سابق له، فإننا نحسب زمن البداية المبكرة للنشاط الحالى بناءً على الوقت المقدر لكل نشاط من الأنشطة السابقة له، ثم نأخذ القيمة الكبرى لها. ففى المثال الحالى نجد أن النشاط رقم ٦ يعتمد على كل من النشاط رقم ٣ والنشاط رقم ٤، ولحساب زمن البداية المبكرة للنشاط رقم ٦ فإننا نحسب الوقت المتوقع لاستكمال كل نشاط سابق له، وهو يساوى ٨ أسابيع للنشاط رقم ٣، ويساوى (١٢) أسبوعاً للنشاط رقم ٤. ثم نأخذ القيمة الكبرى للوقت المتوقع لاستكمال النشاطين وهو (١٢) أسبوعاً ليمثل زمن البداية المبكرة للنشاط رقم ٦. وبالمثل نجد أن النشاط رقم ٧ يعتمد على كل من النشاطين ٥ و ٤ ويكون زمن البداية المبكرة له تساوى (١٢) أسبوعاً أيضاً. وكذلك الحال للأنشطة رقم ٨ و١.

ويعتبر زمن النهاية المبكرة للنشاط الأخير فى المشروع هو تمثيل للفترة الزمنية المطلوبة لاستكمال جميع أنشطة المشروع بدون تأخير وهى تساوى ٢٧ أسبوعاً للمشروع الحالى.

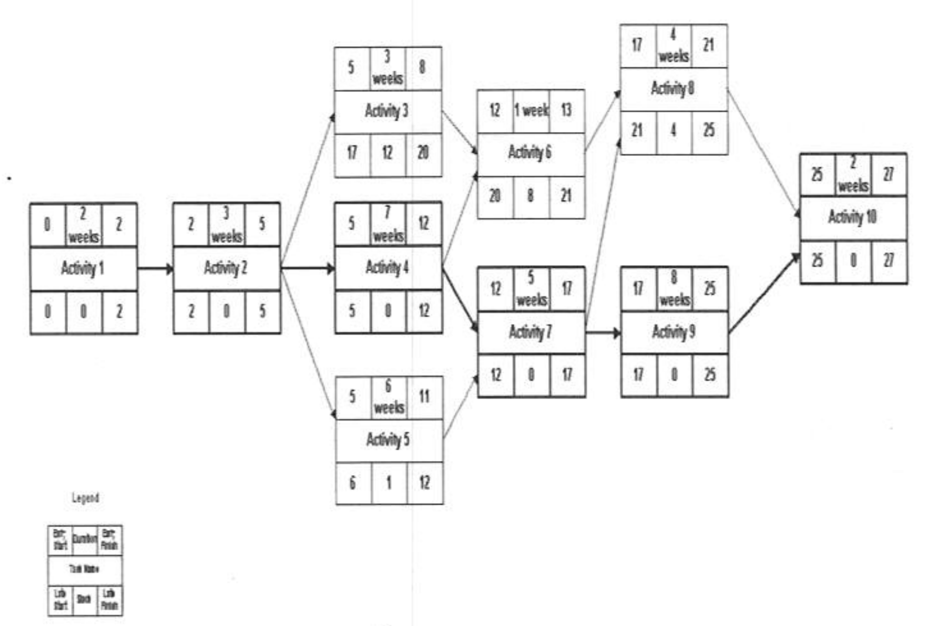
ولحساب زمن البداية المتأخرة لأنشطة المشروع Late Start Time (T)، نبدأ أولاً بالنشاط الأخير فى شبكة المشروع ونقوم بإسناد قيمة زمن البداية المتأخرة للنشاط تساوى قيمة زمن البداية المبكرة له (٢٥ أسبوعاً فى المشروع الحالى). وبعد ذلك نتحرك من اليمين إلى اليسار ونطرح قيمة زمن النشاط الحالى من قيمة زمن البداية المتأخرة للنشاط التالى له فى الترتيب. فمثلاً نجد قيمة زمن البداية المتأخرة للنشاط رقم ٩ = ٢٥ - ٨ = ١٧ أسبوعاً، وقيمة زمن البداية المتأخرة للنشاط رقم ٨= ٢٥ - ٤ = ٢١ أسبوعاً وهكذا. أما النشاط رقم ٧ فإننا نجد أن كلاً من النشاط

رقم ٨ والنشــاط رقم ٩ يتفرعان من النشــاط نفسه. ولحساب زمن البداية المتأخرة له نأخذ القيمة الصغرى لزمن البداية المتأخرة لكل من النشاطين ٨ و ٩ (وهو الزمن الخاص بالنشاط رقم ٩ ويساوى ١٧ أسبوعاً). وعلى ذلك يكون زمن البداية المتأخرة للنشاط رقم ٧ = ١٧ - ٥ = ١٢ أسبوعاً. وهكذا لباقى أنشطة المشروع حتى نصل إلى النشــاط الأول بالمشروع. وأخيراً نحسب قيمة الوقت الفائض لكل نشاط وهو يمثل الفرق بين زمن البداية المتأخرة للنشاط وزمن البداية المبكرة له (Ts = T1-Te).

النشاط الحالى	زمن النشاط	النشاط السابق	زمن البداية المبكرة Te	زمن البداية المتأخرة Tl	الوقت الفائض Ts	المسار الحرج
١	٢	–	٠	٠	٠	√
٢	٣	١	٢	٢	٠	√
٣	٣	٢	٥	١٧	١٢	×
٤	٧	٢	٥	٥	٠	√
٥	٦	٢	٥	٦	١	×
٦	١	٣ و ٤	١٢	٢٠	٨	×
٧	٥	٤ و ٥	١٢	١٢	٠	√
٨	٤	٦ و ٧	١٧	٢١	٤	×
٩	٨	٧	١٧	١٧	٠	√
١٠	٢	٨ و ٩	٢٥	٢٥	٠	√

شكل (٣-٥) حساب قيم الوقت المبكر والمتأخر والفائض لكل نشاط

يوضح شــكل (٣-٥) قيم الوقت المبكــر Te، والوقت المتأخر Tl، والوقت الفائض Ts لكل نشــاط للمثال الحالى. يتميز مخطــط بيرت عن مخطط جانت بأنه يحتوى على معلومات مفيدة فى أثناء عملية مراقبة تنفيذ المشروع مثل الوقت الفائض Slack Time المتاح عند كل نشاط وهو يساعد مدير المشروع فى تحديد الأنشطة الحرجة التــى لا يوجد لديها وقت فائض ومن ثم تحتاج إلى عناية كبيرة فى المتابعة. ويوضح شكل (٣-٦) كيفية تمثيل أنشطة المشروع الحالى باستخدام مخطط بيرت Pert.



شكل (٣-٦) مخطط بيرت Pert لأحد المشاريع

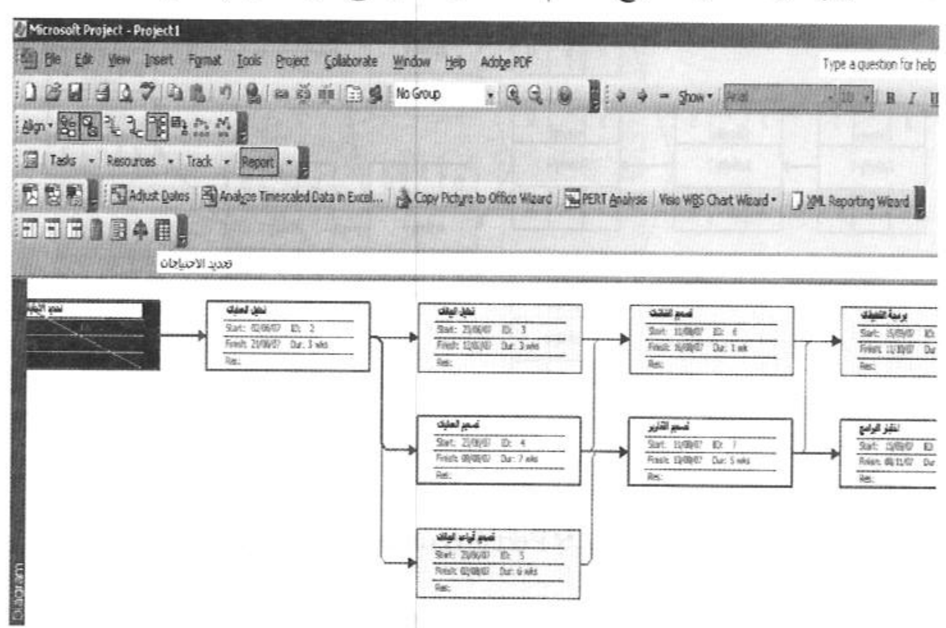
٣-١-١-٣ طريقة المسار الحرج Critical Path Method:

يعرف المسار الحرج داخل مخطط بيرت بأنه المسار الذى يمثل أقصر وقت يمكن أن يتم تنفيذ المشــروع خلاله. يتم تحديد المسار الحرج من خلال أنشطة المشروع التى لا تحتوى على وقت فائض (Ts=٠)، وتعرف الأنشــطة التى يمر خلالها المســار الحرج بأنها أنشــطة حرجة Critical Activities، بمعنى أن أى تأخير يحدث فى أى نشــاط موجود على المســار الحرج يؤدى بالقطع إلى تأخير المشروع كله. وبالعكس فإن أى نشــاط لا يقع على المســار الحرج يحتوى على وقت فائض (Ts>٠) مما يعنى أن التأخير فى تنفيذ هذا النشــاط لا يؤدى إلى تأخير المشــروع كله. ويعرف الوقت الفائض لأى نشــاط بأنه الوقت المسموح للنشــاط بالتأخير من دون التأثير فى وقت اكتمال المشروع.

يمكننا حساب الوقت الفائض لأى نشاط كما يلى:

الوقت الفائض (Ts) = زمن البداية المتأخرة للنشاط (Tl) – زمن البداية المبكرة للنشاط (Te).

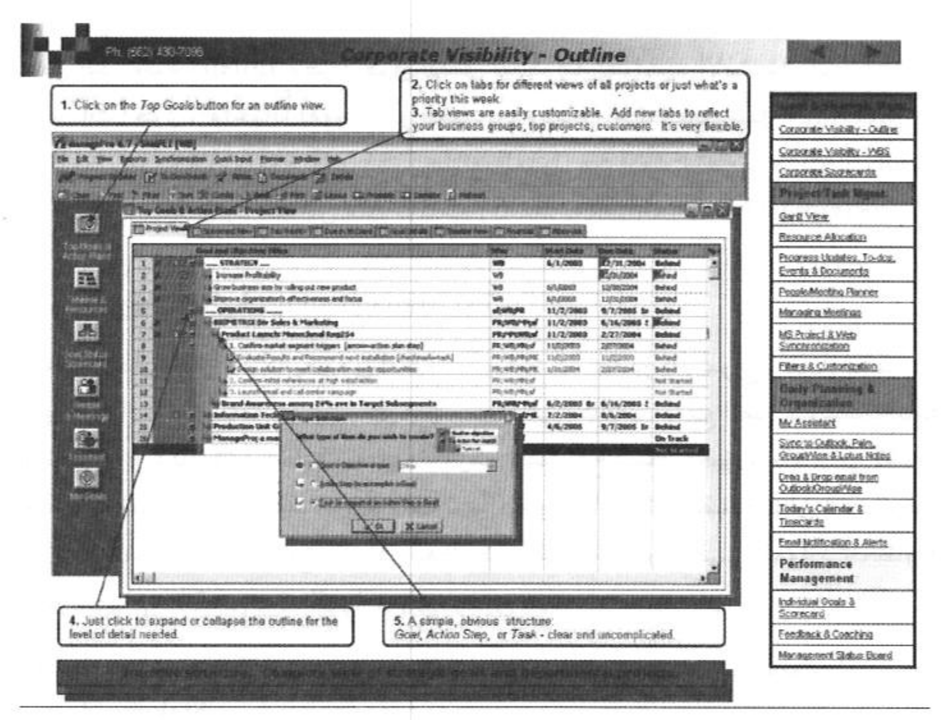
يوضح شكل (٣-٧) مثالاً على استخدام برنامج Microsoft Project لإعداد مخطط بيرت والمسار الحرج لنظام الأعمال الموضح فى شكل (٣-٢).



شكل (٣-٧) استخدام برنامج Microsoft Project **لإعداد مخطط بيرت**

٣-١-٢ إدارة الأداء Performance Management:

من الضرورى لأى مؤسسة تجارية تريد المنافسة بنجاح فى سوق العمل أن تقوم بمراجعة وتعديل أهدافها وخططها تبعاً للتغيرات التى تحدث فى متطلبات العميل أو نتيجة استحداث تقنيات جديدة مفيدة لنظام الأعمال أو نتيجة حدوث تغييرات فى البيئة الخارجية للمشروع تؤثر فيه. كل ذلك يتطلب من مدير المشروع أن يركز على متطلبات الكفاءة وتحسين الأداء للأفراد القائمين على تطوير المشروع. هناك بعض الأدوات التى تدعم العديد من الخدمات والأنشطة الخاصة بإدارة المشروع ومن ضمنها خدمة إدارة الأداء والتخطيط الإستراتيجى التى تساعد الأفراد وفرق العمل على تخطيط ومتابعة عملية إنجاز الأعمال المكلفين بها. يوضح الشكل (٣-٨) مثالاً لاستخدام أداة ManagePro التى تقدم خدمة إدارة الأداء وخدمات أخرى مثل إدارة الاجتماعات والرسائل الإلكترونية وإنتاج الوثائق، بالإضافة إلى الخدمات الأساسية لإدارة المشروع مثل إدارة الأهداف وتطوير خطة المشروع وجدولة المهام وإدارة الموارد.



شكل (٣-٨) أداة ManagePro لإدارة الأداء

٣-١-٣ إدارة التعاون Collaboration Management:

أحياناً يتم توزيع مهام مشــروع البرمجيات على فرق عمل تعمل فى مواقع مختلفة ومتباعدة فى الوقت نفســه ويتم الاتصال بين الأفراد المشاركين فى عمليات التطوير من خلال شــبكة الإنترنت. وهنا يقوم مدير المشروع بإدارة عملية التعاون بين أفراد فرق العمل الموزعة عن بعد باســتخدام تقنية شــبكة الإنترنت. فيقوم أولاً بتخطيط عملية الاتصال والتعاون بين فرق العمل وتخصيص المهام والمسؤوليات وأيضاً اتخاذ القرارات من خلال الشبكة. ومن ثم يقوم الأفراد المشاركون فى فرق العمل باستلام المهام وتســليم منجزاتهم وملاحظاتهم أو آرائهم لمدير المشروع عبر شبكة الإنترنت أيضاً. ويقوم مدير المشــروع باستلام جميع المنجزات والمقترحات وتقييمها وإبداء الملاحظــات عليها وإجراء التكامل المطلوب بيــن المنتجات، ثم يقوم بإصدار تقرير بالمنجزات وينشــره على صفحات الويب الخاصة بالمشــروع ليتم التعرف عليها من

قبل جميع المشـــاركين والمهتمين بالمشـــروع. ويقوم مدير المشـــروع أيضاً بإرسال رسائل إلكترونية Emails لأفراد فرق العمل ليذكرهم بمواعيد الأنشطة الموزعة وأى معلومـــات أخرى. وأحياناً يقوم مدير المشـــروع بعقد اجتماعات عمل Meetings مع فرق العمل من خلال شبكة الإنترنت. هناك العديد من أدوات تطوير صفحات الويب التى تدعم عملية إدارة التعاون مثل برنامج (www.eroom.net, 2006) eRoom.

٣-١-٤ إدارة التكاليف Cost Management:

تعتبر عملية إدارة التكاليف من المهام الأساســية لمدير المشروع. فمن المعروف أن ميزانية المشـــروع تعتمد بشـــكل أساســـى على تكاليف جميع الموارد المستخدمة بالمشروع. ولذلك فإن عملية تحديد تكاليف المشروع بشكل دقيق تساهم فى تقدير ميزانية المشروع بشكل جيد. وبعد ذلك تبدأ عملية إدارة التكاليف لضمان عدم تجاوز الميزانية المقدرة للمشـــروع. تتركز معظم تكاليف المشـــروع فـــى الموارد المطلوبة لإنجاز المهام. تنقسم هذه الموارد إلى نوعين هما:

١- مـــوارد بشـــرية: تتضمـــن جميع الأفراد المشـــاركين فى عمليات تطوير المشـــروع Stakeholders.

٢- مـــوارد ماديـــة: تتضمن جميـــع الأجهـــزة والبرمجيات والتجهيـــزات ومصاريف الاستهلاك وغيرها.

ويتم تصنيف التكاليف بالمشروع إلى صنفين أساسيين هما:

١- تكاليف اســـتثمارية: هى تكاليف يتم اســـتثمارها قبل بداية المشروع مثل شراء أو تأجير الموقع الخاص بالمشـــروع وتكاليف تجهيز الموقع بجميع الموارد المطلوبة لبدء المشروع.

٢- تكاليف تشـــغيلية: هى تكاليف تنشـــأ مع بدء التشغيل الفعلى للمشروع مثل إيجار أجهزة وأجور الموارد البشرية وتكاليف استهلاك خدمات مثل الكهرباء والاتصالات وغيرها.

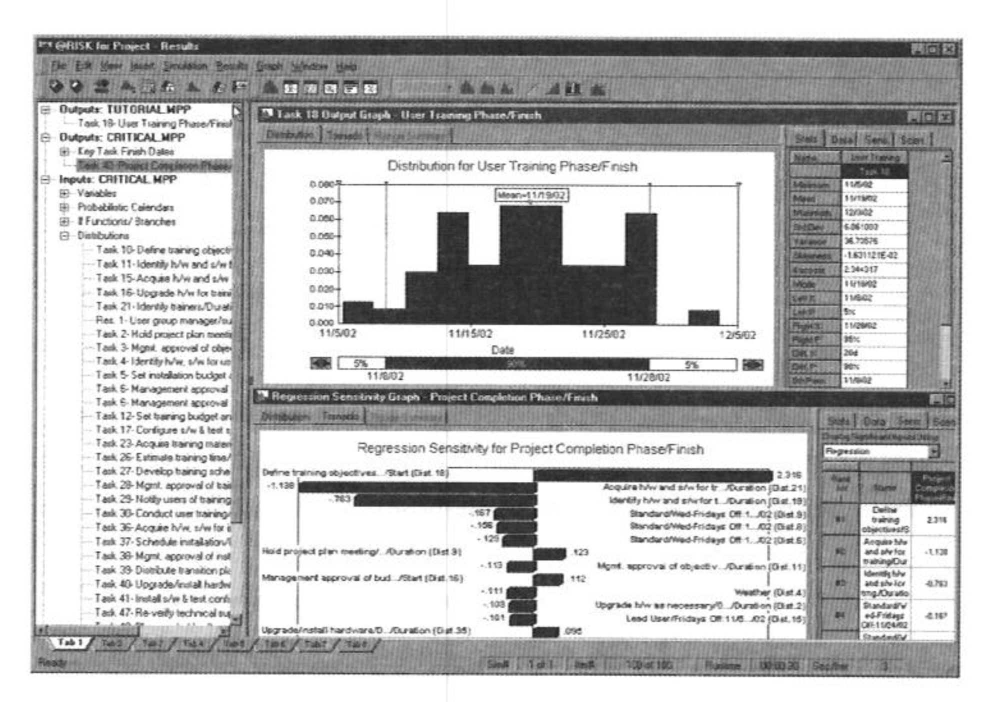
وتتضمن عملية إدارة التكاليف ما يلى:

- تحديد الموارد المطلوبة بدقة.
- تقدير تكلفة الموارد.
- تخصيص الموارد.
- متابعة عملية استخدام الموارد.
- التحكم فى الاستهلاك.
- تقييم الموارد المستخدمة.

هناك عدة طرق تستخدم لتقدير تكاليف المشروع وسوف نستعرضها فى الفصل القادم. وهناك أيضاً العديد من الأدوات التى تستخدم فى عملية إدارة التكاليف (ضمن عمليات إدارة المشروع) مثل برنامجى Primavera Enterprise (Primavera،2006)، eProject (eProject، 2006).

٣-١-٥ إدارة المخاطر Risk Management:

تعرف المخاطر بأنها مجموعة الظروف المؤثرة سلبياً فى عمليات تطوير المشروع ومن ثم تؤثر فى جودة المنتج البرمجى. تحتوى جميع أنشطة المشروع على قدر من المخاطر حسب طبيعة النشاط من حيث الوضوح أو الصعوبة والموارد المستخدمة. ولذلك فإن نشاط إدارة المخاطر يعتبر من الأنشطة المهمة فى إدارة المشروع، حيث يقوم مدير المشروع بتحديد الأنشطة التى يمكن أن تحتوى على مخاطرة ويقوم أيضاً بتقييم قيمة المخاطر وإعداد خطة للتغلب عليها عند حدوثها. يستخدم مدير المشروع أساليب الاحتمالات Probability والمحاكاة Simulation لتقدير قيمة المخاطر المتوقعة التى تساعده على اتخاذ القرار الصحيح. هناك بعض الأدوات التى تستخدم بكثرة فى أثناء عملية تقدير المخاطر. فمثلاً يتم استخدام أدوات الجداول الإلكترونية Spreadsheets مثل Microsoft Excel وأدوات قواعد البيانات Databases مثل Microsoft Access، وأدوات المحاكاة Simulation مثل Monte Carlo، بالإضافة إلى أدوات إدارة المشروع فى أثناء عمليات تقدير المخاطر المحتملة بالمشروع. يوضح شكل (٣-٩) مثالاً لاستخدام مخرجات برنامج Monte Carlo من خلال أداة Risk (Palisade, 2003).



شكل (٣-٩) استخدام أداة Risk فى تقدير المخاطر

٣-٢ أدوات نمذجة النظام System Modeling Tools:

كما ذكرنا سابقاً فإن عملية تطوير مشروع البرمجيات تعتمد على أسلوب النمذجة Modeling فى معظم أنشطة التطوير، ويتطلب ذلك توافر مجموعة من الأدوات التى تدعم هذا الأسلوب. هناك مجموعة من الأدوات التى تدعم أسلوب النمذجة المرئية Visual Modeling خلال المراحل المختلفة لعملية تطوير أنظمة البرمجيات. أحياناً يطلق على هذه الأدوات اسم «أدوات هندسة البرمجيات المدعمة بالحاسب CASE" وهى تتميز بدعمها لأســلوب المخزن العام لبيانات النظام أو المستودع Repository الذى يحتوى على جميع المعلومات عن نظام الأعمال، ومن ضمنها النماذج التى تصف النظام من خلال عدة منظورات مختلفة. يســاعد هذا المســتودع على إمداد جميع أفــراد فريق التطوير بالمعلومات المطلوبة. تركز أدوات نمذجة النظم على أنشــطة تحليل المتطلبات Requirements Analysis وتصميم النظام System Design. فيما يلى عرض أمثلة لبعض هذه الأدوات:

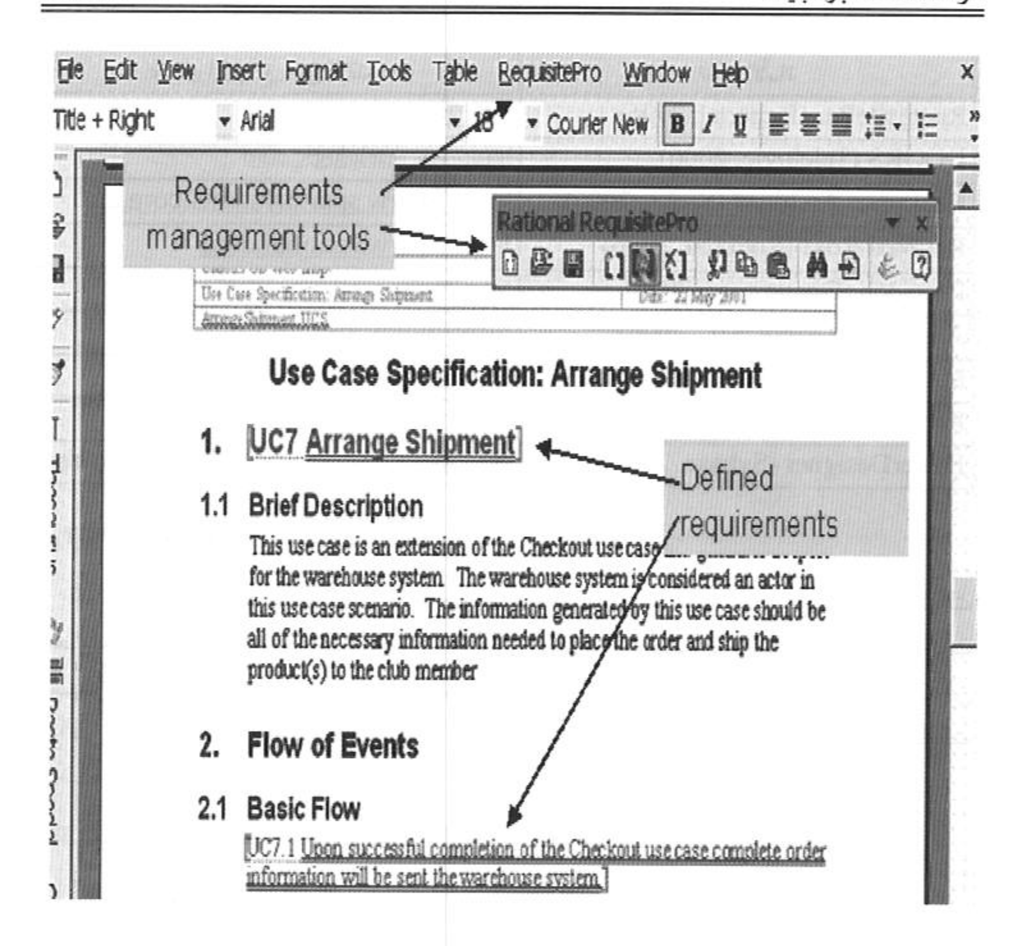
عنوان الموقع على الإنترنت	اسم الأداة
http://www.rational.com	RequisitePro (Rational, 2006)
http://www.telelogic.com	DOORS (Telelogic, 2006)
http://www.sparxsystems.com.au/	Enterprise Architect (Sparx, 2006)
http://www.rational.com	Rational Rose (Rational, 2006)
http://www.gentleware.com	Poseidon (Gentleware, 2006)
http://www.argouml.tigris.org	ArgoUML (ArgoUML, 2006)
http://www.magicdraw.com	MagicDraw (Nomagic, 2006)
http://www.rational.com	Rational SODA (Rational, 2006)
http://www.sybase.com/products/enterprisemodeling	PowerDesigner (Sybase, 2006)

٣-٢-١ إدارة المتطلبات Managing Requirements:

المتطلبات هي عبارة عن مجموعة من الجمل النصية التى تصف متطلبات واحتياجات العميل الخاصة بنظام الأعمال الجارى تحليله. تتبع وثائق المتطلبات تنسيقاً وشكلاً معيناً يلائم التمثيل الهرمى Hierarchical Representation. يجب أن تتوافر مجموعة من الخصائص فى الأداة التى سوف تستخدم فى نمذجة متطلبات النظام مثل:

- تقديم نموذج قياسى يلائم وصف المتطلبات.
- دعم أسلوب التمثيل الهرمى.
- إدارة عملية تغيير المتطلبات.
- إتاحة بيانات المتطلبات لجميع أفراد المشاركين فى عمليات التطوير.

يستخدم فريق التحليل هذه المتطلبات لإنتاج نماذج تحليل النظام. ويستخدم فريق التصميم مواصفات المتطلبات لتحديد معمارية النظام. ويستخدم فريق الاختبار هذه المواصفات لإعداد حالات الاختبار للمنتجات البرمجية. ويستخدمها مدير المشروع لإعداد جدول أنشطة المشروع وتحديد الموارد المطلوبة ومن ثم تقدير ميزانية المشروع. يوضح شكل (٣-١٠) جزءاً من وثيقة المتطلبات لإحدى حالات الاستخدام use case التى تم إنشاؤها باستخدام إحدى أدوات إدارة المتطلبات المقدمة من شركة IBM ويطلق عليها اسم RequisitePro (Rational, 2006).



شكل (٣-١٠) وثيقة المتطلبات

يتم تخزين وثيقة المتطلبات فى مستودع النظام، وتقوم أداة إدارة المتطلبات بعرض تلك المتطلبات لأفراد فريق التطوير وتسمح لهم أيضاً بالتعديل فيها حسب الحاجة. تدعم الأداة أيضاً عمليات التنقل بين المتطلبات وبين النماذج المرئية المقابلة لها. وتستخدم أدوات النمذجة أسلوب الجداول الإلكترونية لعرض المتطلبات وتداولها.

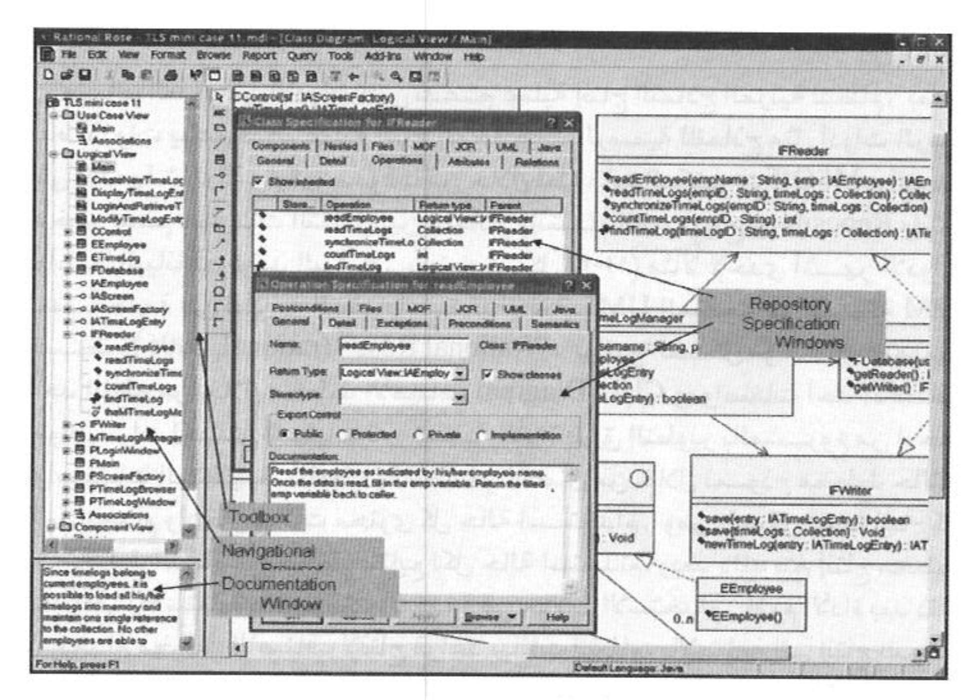
تهتم عملية إدارة المتطلبات بتنظيم عملية ربط المتطلبات بالنموذج الذى يقوم بتمثيلها وبالعناصر المختلفة داخل النموذج مثل الأصناف Classes، والطرق Methods. يؤدى ذلك إلى توافر إمكانية التكامل بين مراحل التطوير المختلفة بدءاً من تحديد المتطلبات مروراً بالتصميم وحتى إنتاج البرامج.

٣-٢-٢ النمذجة المرئية Visual Modeling:

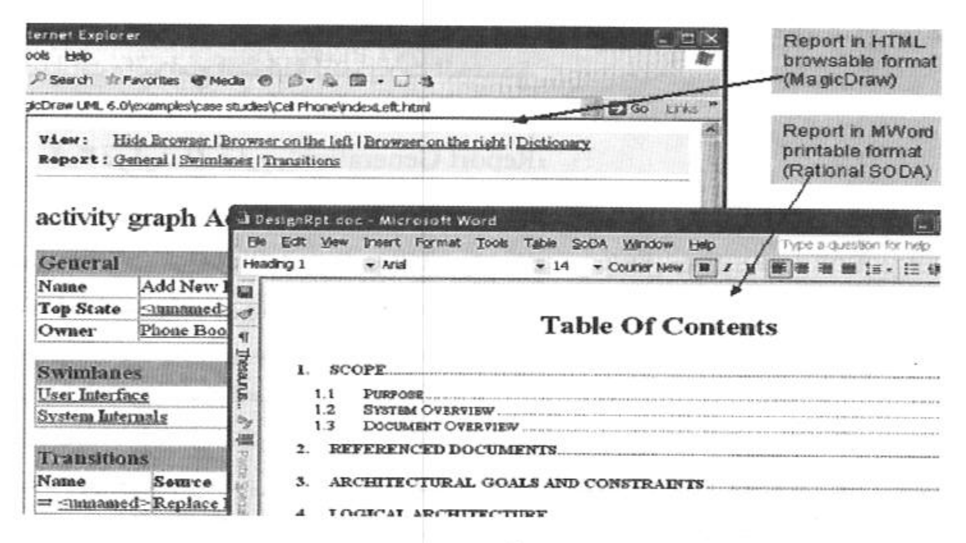
هنــاك العديد من الأدوات التى تدعــم عملية إنتاج النماذج المرئية للنظام. بعض هذه الأدوات يدعم فقط عملية إنتاج المخططات الرسمية للنماذج مثل أدوات الرسم التى تقدمها شــركة مايكروســوفت من خلال تطبيقات المكتب MSOffice، والبعض الآخــر يدعم عمليــات التكامل بين النماذج وإنشــاء المســتودع Repository وإنتاج قواعــد البيانات وتوليد البرامج. يوضح شــكل (٣-١١) مثالاً لإحدى أشــهر الأدوات المســتخدمة فى نمذجة النظم باســتخدام لغة UML المقدمة من شــركة IBM وتســمى "Rational Rose" (rational, 2006). يوضح الشــكل مواصفات التصميم لأحد النظم من خلال مخطط الأصناف Class Diagram ومواصفات أحد الأصناف ومواصفــات إحــدى العمليات له. تمكــن الأداة فريق التطوير بالمشــروع من إدخال مواصفــات المتطلبــات الوظيفية لنظــام الأعمال من خلال نمــوذج مخطط حالات الاســتخدام وإدخال بيانات محتوى كل حالة اســتخدام. ومن الممكن أيضاً إنتاج كل من مخطط النشاط ومخطط التتابع لكل حالة استخدام، وبعد ذلك يتم إنتاج مخطط الأصناف بالاستعانة بالمعلومات المقدمة فى حالات الاستخدام. تقوم الأداة بعد ذلك باستخدام مخطط الأصناف لإنتاج قواعد بيانات النظام، بالإضافة إلى إنتاج البرامج المبنية على الأصناف بالاســتعانة بالمعلومات الموجودة فى مخططات التتابع لوصف التفاعــلات بين الأصناف. تمكــن الأداة أيضاً فريق التطوير مــن إجراء التعديلات المناسبة على النماذج المخزنة بالمستودع فى أى وقت مع إمكانية إعادة إنتاج قواعد البيانات والبرامج المقابلة للتعديلات الجديدة.

٣-٢-٣ توليد التقارير Report Generation:

من الضرورى أن يتم إمداد باقى المشاركين فى المشروع Stakeholders بمعلومات عن منجزات مراحل التطوير المختلفة. يتم ذلك من خلال مجموعة من التقارير التى يتم إعدادها بشــكل يناسب المســتخدم للتقرير مثل وثائق نصية أو صفحات ويب أو أشــكال توضيحية ورسوم بيانية وغيرها. يوجد العديد من الأدوات التى تدعم عملية توليد التقارير وهى تحتوى على عدد كبير من القوالب Templates يمكن اســتخدامها فى تحديد شــكل التقرير. يحدد القالب الشــكل العام لهيكل التقرير المطلوب، وبعد ذلك يتم تحديد محتوى التقرير حســب حاجة المســتخدم من المعلومات عن النظام. يوضح شــكل (٣-١٢) اســتخدام إحدى أدوات النمذجة وإنتاج التقارير فى أشــكال مختلفة (Rational, 2006).



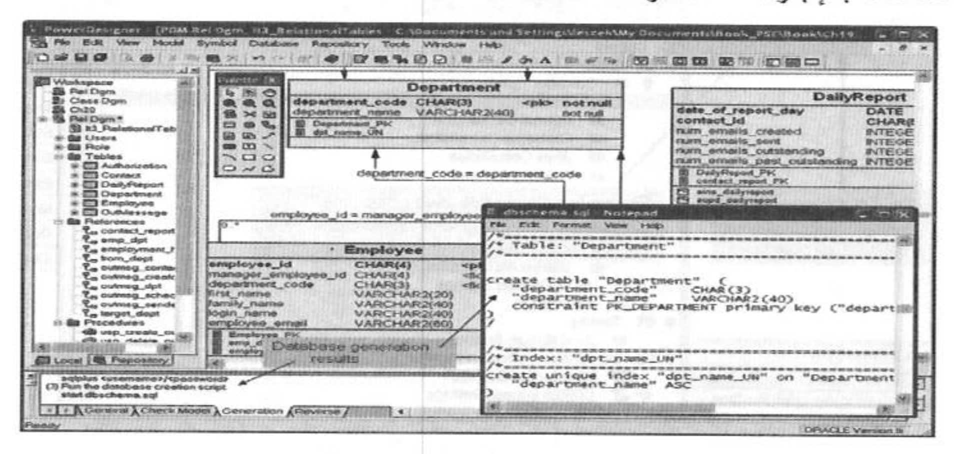
شكل (٣-١١) استخدام أداة النمذجة المرئية Rational Rose



شكل (٣-١٢) استخدام أدوات النمذجة لإنتاج التقارير

٣-٢-٤ نمذجة قواعد البيانات Database Modeling:

كما ذكرنا سابقاً فإن جميع بيانات النظام يتم تنظيمها وتخزينها والتعامل معها باستخدام أحد أنظمة إدارة قواعد البيانات. تحتوي قواعد بيانات النظام على مجموعة من الجداول المرتبطة، ويمثل كل جدول كائناً من كائنات النظام أو يمثل علاقة تربط بين بعض الكائنات. يتم تمثيل ذلك من خلال نماذج قواعد البيانات التى تصف مجموعة الكائنات بالنظام والعلاقات بينها. هناك العديد من الأدوات التى تدعم عملية نمذجة قواعد بيانات النظام. تقوم هذه الأدوات بعمليات إنشاء الجداول Creating Tables وتحديد خصائص Attributes كل جدول والمفاتيح الأساسية Primary Keys والخارجية Foreign Keys له، وكذلك مجموعة القيود Constraints والإجراءات المخزنة Stored Procedures بكل جدول، بالإضافة إلى تمثيل العلاقات Relationships بين الجداول. من الممكن أن تستخدم بعض هذه الأدوات نموذج الأصناف Class Diagram بلغة UML لإنتاج نموذج قواعد البيانات، ثم يضاف إليه بعض الخصائص مثل قيود السلامة المرجعية Referential Integrity Constraints. ومن الممكن أيضاً أن يتم استخراج نماذج لغة UML من نموذج قواعد البيانات باستخدام أسلوب الهندسة العكسية Reverse Engineering. يوضح شكل (٣-١٣) مثالاً لاستخدام أداة Power Designer المقدمة من شركة Sybase لنمذجة قواعد البيانات لأحد النظم (Sybase, 2006). يوضح الشكل بعض الجداول والعلاقات بينها. والجدير بالذكر أن أداة نمذجة قواعد البيانات تعمل بالتكامل مع أجزاء أخرى داخل نظام إدارة قواعد البيانات Database Management System وهى تتسم بقدرتها على توليد أكواد هيكل قواعد البيانات schema code والأكواد الخاصة بالإجراءات المخزنة Stored Procedures.



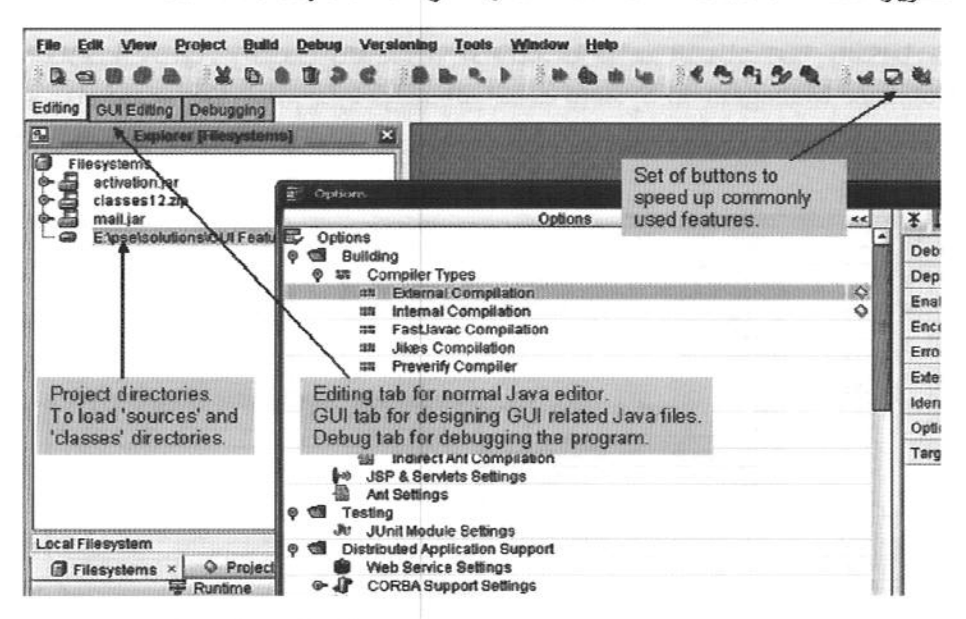
شكل (٣-١٣) استخدام أداة Power Designer **لنمذجة قواعد البيانات**

٣-٣ بيئة التطوير المتكاملة (Integrated Development Environment (IDE:

المقصـــود ببيئة التطوير المتكاملـــة هى البيئة التى تســـاعد المبرمج على إنجاز مهـــام كتابة البرامج وتدقيقها وتكاملها مع أجزاء برمجية أخرى وإنتاج البرمجيات فى الشـــكل التنفيذى. هناك العديد من الأدوات التى تدعم هذا الأسلوب ولكنها تختلف فى بعض الخصائص أو المميزات التى تقدمها للمطورين مثل دعمها لأسلوب نمذجة النظم البرمجية وتكاملها مع أدوات أخرى مثل أدوات إدارة التشـــكيل Configuration Management Tools لدعم أسلوب فريق العمل أثناء عملية التطوير.

٣-٣-١ كتابة وتنفيذ وتصحيح البرامج Writing, Executing, and Debugging Programs:

الغرض الأساسى من استخدام بيئة التطوير المتكاملة هو مساعدة المبرمجين فى كتابة الأكواد بشـــكل أسرع وتقليل نسبة الخطأ فيها وذلك من خلال توفير بيئة تطوير تدعم أسلوب البرمجة المرئية والبرمجة الموجهة بالكائنات وتكاملها مع كل من نماذج لغة UML ونظم إدارة قواعد البيانات. يوضح شـــكل (٣-١٤) مثالاً لاســـتخدام بيئة تطوير متكاملة Sun ONE Studio تقدمها شركة Sun Microsystems.



شكل (٣-١٤) بيئة التطوير المتكاملة Sun ONE Studio

تدعم أدوات بيئة التطوير المتكاملة أسلوب البرمجة الموجهة بالكائنات من خلال تقديمها لعدد كبير من الكائنات المعدة سابقاً والجاهزة لإعادة الاستخدام Reusable Objects فى التطبيقات التى يتـم تطويرها مع إمكانية ضبطها والتعديل فيها لتلائم طبيعــة التطبيق. توفر أيضــاً هذه الأدوات مجموعة من المعالجــات wizards التى تســاعد المبرمج فى إنتاج البرمجيات بشــكل أســرع ودقيق. وتتميــز بيئة التطوير المتكاملة أيضاً باحتوائها على محرر نصوص حســاس للكائنات المســتخدمة، يقوم باســتعراض خصائصها والطرق التى تقدمها مما يسهل من عملية كتابة الأكواد بشكل صحيــح وســريع. وتقوم أيضاً بتقديم المســاعدة للمبرمج فــى حالة حدوث أخطاء أثناء التعامل مع الكائنات المســتخدمة فى التطبيــق. وتدعم بيئة التطوير المتكاملة عملية تنفيذ البرامج بعد تدقيقها وترجمتها وربطها بالمكونات المستخدمة من مكتبة الكائنات حتى يتم إصدار الشكل التنفيذى للتطبيق. وتدعم أيضاً عملية تتبع الأخطاء وتصحيحهــا Debugging من خلال التنفيذ الجزئى للبرنامج حســب رغبة المبرمج (خطــوة خطوة أو حتى نقطة معينــة أو تنفيذ طريقة معينة Method أو تخطى تنفيذ طريقة معينة فى أثناء التنفيذ).

٣-٣-٢ التكامل مع نمذجة البرمجيات Integration with Software Modeling:

تتميــز بعــض أدوات التطويــر الحديثة بتضمينهــا عمليات نمذجــة البرمجيات باســتخدام لغة UML فى بيئة تطوير تناســب أســلوب فريق العمــل، إذ تمكنهم من التعاون والمشــاركة الفعالة فى تطوير النماذج المختلفة للنظام البرمجى واســتخراج الأكواد وقواعد البيانات المقابلة لها. ويمكن للمطورين إجراء التعديلات فى النماذج والحصول على الأكواد المقابلة آلياً. ويمكن أيضاً استخدام أسلوب الهندسة العكسية فى استخراج النماذج من الأكواد المكتوبة وقواعد البيانات.

٣-٣-٣ تطوير تطبيقات الأعمال Enterprise Application Development:

تســتخدم أدوات بيئة التطوير المتكاملة IDE فى تصميم وتطوير ونشر تطبيقات نظم الأعمال الكبيرة والمتوسطة أيضاً. تتطلب عملية التطوير أفراداً ذوى كفاءة وخبرة عاليــة فى مجالات التصميــم والبرمجة وقواعد البيانات، مــع الإلمام التام بمفاهيم الكائنــات الموجهة واســتخدامها فى مراحل التطوير المختلفــة. وكذلك التمكن من اســتخدام أدوات تطوير صفحات الإنترنت، واستخدام تقنية الخادم والعميل /Client Server. تتميز هذه الأدوات أيضاً بإمكانية التكامل مع خادم التطبيقات Application

Server. تشترك معظم أدوات تطوير تطبيقات نظم الأعمال فى تنفيذ ما يلى:

- تصميم المكونات الأساسية للتطبيق (نماذج Forms أو صفحات ويب Web Pages).
- تطوير محتوى المكونات باستخدام المعالجات المساعدة Wizards.
- إضافة أكواد ضرورية للمكونات المنتجة باستخدام المعالجات.
- اختبار المكونات.
- نشر التطبيق.

هناك العديد من الأدوات التى تستخدم على نطاق واسع فى تطوير تطبيقات نظم الأعمال مثل:

- Oracle JDeveloper, 10g.
- Sun ONE Studio.
- IBM San Francisco.
- Microsoft Visual Studio.

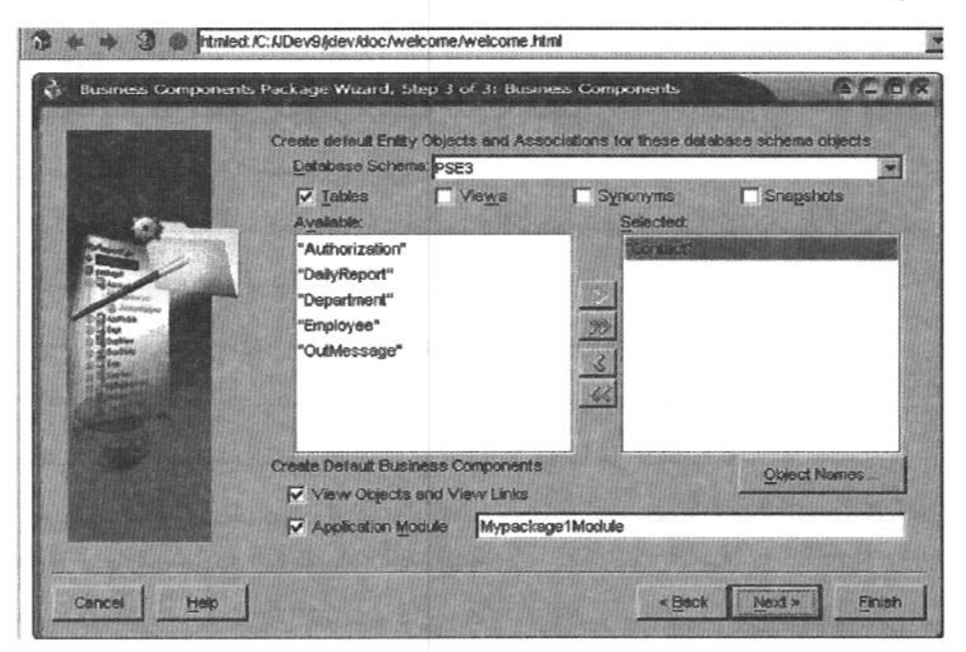
تحتوى معظم هذه الأدوات على معالجات قوية تساعد المطورين على إنجاز جزء كبير من التطبيق بسهولة وسرعة كبيرة. وتقدم أيضاً هذه الأدوات بعض القوالب الجاهزة Templates مثل JSP, EJB, ASP, Web services وهى تختصر كثيراً من الوقت والجهد المطلوب لتطوير التطبيق. تدعم أيضاً هذه الأدوات الاحتياجات الضرورية لتطبيقات الأعمال مثل:

- استخدام أسلوب النمذجة المرئية Visual Modeling.
- الاتصال بقواعد البيانات Database Connectivity.
- تقديم أنماط مختلفة لواجهات المستخدم User Interface Formats.
- دعم عمليات الهندسة العكسية Reverse Engineering Support.
- التعامل مع خادم التطبيقات Application Server.
- إمكانية العمل فى بيئة الإنترنت Web Based/Enabled Development.

٣-٣-٤ التكامل مع مكونات نظام الأعمال Integration with Business Components:

تعتبر مكونات نظام الأعمال Business Components هى الطبقة المتوسطة Middle- Tier داخل نظام البرمجيات المتعدد الطبقات Multi-Tier System. توجد هذه الطبقة بين طبقة واجهة التطبيق Application GUI وطبقة قواعد بيانات التطبيق Application Database. ومن الأفضل أن يتم نشر هذه الطبقة المتوسطة

فى خادم التطبيقات Application Server. تعتبر أداة التطوير Oracle JDeveloper مـن الأدوات التى تدعم أسـلوب طبقة مكونات نظام الأعمال الذى يطلق عليه اسـم «Business component for Java (BC4J)» الـذى يسـاعد علـى تطوير التطبيقات المبنيـة على قواعد بيانات. تحتـوى طبقة BC4J على كتل بنائية Building Blocks تتكامل مع قواعد البيانات باستخدام أسلوب الاتصال JDBC. وتتضمن طبقة BC4J أيضاً مجموعة ملفات تشـكيل مبنية على أسـلوب XML تقوم بعملية الربط بقواعد البيانـات وإنتاج نوافذ التطبيـق Application Windows. تسـتخدم طبقة BC4J تقنيات JDBC, SQLJ للاتصال بقواعد البيانات واستخراج البيانات المطلوبة لواجهة التطبيق. يوضح شكل (٣-١٥) إحدى خطوات المعالج الخاص بطبقة BC4J فى أثناء الاتصال بقواعد البيانات فى أثناء استخدام أداة تطوير التطبيقات المبنية على قواعد البيانـات Oracle JDeveloper المقدمة من شـركة أوراكل. يتميز أسـلوب المعالج بتبسـيط عمليات الاتصال بقواعـد البيانات واختيار الجداول التى يتم اسـتخدامها فى بناء النموذج الخاص بالتعامل مع البيانات، وكذلك تكوين الاسـتعلامات المناسبة لإدخال واستخراج البيانات من قواعد البيانات، مما يوفر الكثير من الوقت والمجهود لمطورى التطبيقات.



شكل (٣-١٥) استخدام أداة التطوير Oracle JDeveloper

٣-٤ أدوات هندسة البرمجيات المدعمة بالحاسب CASE Tools:

لقد عانت مشـــاريع البرمجيات فى الماضى القريب من اعتماد عملية التطوير على أســلوب ومهارات الأفراد القائمين على تنفيذها وهى تتباين بشــكل كبير من شــخص لآخر، مما يؤدى إلى صعوبة شديدة فى تكامل الأنظمة وصيانتها فيما بعد، نظراً لعدم تقيدهـا أو اتباعها لمقاييس علمية أو منهجية معينة. وللتغلب على تلك المشــكلة؛ تم إنتاج بعض الأدوات التى تســتخدم الأساليب القياسية لهندسة البرمجيات وذلك لدعم عملية تطوير النظم. تفضل الآن أغلب المؤسسات استخدام أدوات هندسة البرمجيات المدعمة بالحاسب فى تطوير وإدارة أنظمتها لتحقيق العديد من الأغراض مثل:

- تحسين جودة النظام المطور.
- إنجاز النظم فى وقت أقل من الطرق التقليدية.
- تسهيل وتحسين عملية الاختبار باستخدام أسلوب التدقيق الآلى للبيانات.
- تحسين عملية التكامل بين أنشطة المشروع.
- تحسين جودة عملية توثيق النظام وضمان اكتمالها.
- تسهيل عمليات الصيانة للنظام.
- المساعدة فى اعتماد معايير قياسية لعملية التطوير.
- المساعدة فى إدارة المشروع.

وتتمثل معوقات استخدام أدوات هندسة البرمجيات المدعمة بالحاسب فيما يلى:

- التكلفة العالية لشراء تلك الأدوات.
- التكلفة العالية لتدريب فريق التطوير على استخدامها.
- ضعف درجة الثقة فى منتجاتها من قبل بعض المؤسسات.
- عدم اتباع المعايير القياسية للمنهجيات المتبعة داخل المؤسسة.

ويتمثل تأثير استخدام أدوات هندسة البرمجيات على جميع الأفراد المشاركين فى تطوير وإدارة مشاريع البرمجيات فيما يلى:

- تساعد أفراد التحليل والتصميم على إنتاج جزء كبير من المهام الروتينية بشكل آلى بناءً على المواصفات الأساسية التى تم إدخالها، مما يوفر الوقت والمجهود.

- تســـاعد كثيراً أفــراد فريق البرمجة، إذ إنها تقوم بإنتـــاج الأكواد المقابلة لمواصفات التصميـــم، ويقتصر دور المبرمجين على تكامل وصيانـــة المكونات البرمجية المنتجة وإضافة بعض التحسينات الشكلية عليها.
- تساعد مستخدمى النظام ليصبحوا أكثر فاعلية مع مطورى النظام فى أثناء عمليات تطوير البرمجيات المطلوبة ومراجعتها وإبداء الملاحظات عليها.
- تســـاعد العميل (صاحب المشـــروع) ليكون أكثر فاعلية مع مدير المشـــروع فى أثناء عمليات التخطيط الإستراتيجى لمشروع الأعمال.
- تســـاعد قائد كل فريق من فرق التطوير بالمشـــروع على القيام بدور أكبر فى قيادة ومتابعة إنتاج كل فرد من أفراد الفريق وكذلك عملية التكامل بين منتجات الفريق.
- تساعد مدير المشروع على إدارة المشروع بشكل أفضل ومتابعة تقدم العمل بالمشروع أولاً بـــأول والتحكم فى المتغيرات والموارد الخاصة بالمشـــروع، مما يســـاهم فى رفع درجة نجاح مشروع الأعمال فى تحقيق أهدافه.

مكونات أدوات هندسة البرمجيات المدعمة بالحاسب:

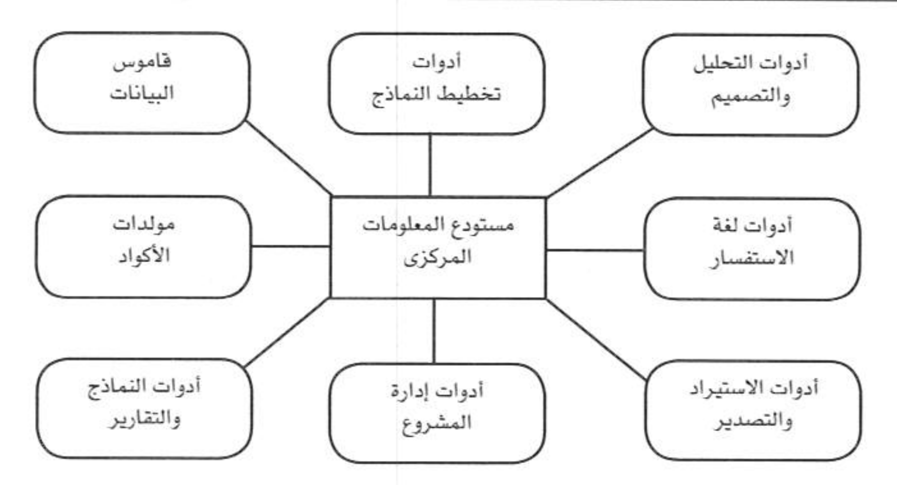
تدعم أدوات هندســـة البرمجيات معظم أنشطة تطوير وإدارة مشاريع البرمجيات، فهناك أدوات تســـتخدم فى مرحلة تعريف المشروع واختياره، وهناك أدوات تستخدم فى أثناء مرحلة التخطيط، وهناك أدوات تستخدم لإنتاج النماذج التخطيطية لكل من عمليات النظام وبياناته، وكذلك عملية توليد البرامج، وإنتاج نماذج واجهات استخدام النظام والتقارير، بالإضافة إلى التوثيق الآلى لجميع مراحل التطوير. يمكننا تنظيم تلك الأدوات فى شكل طبقتين: تسمى الطبقة الأولى «أدوات الطبقة العليا Upper CASE Tools"، وتســـمى الطبقـــة الثانية «أدوات الطبقة الســـفلى Lower CASE Tools"، كمـــا هو مبين بالشـــكل (٣-١٦). تحتـــوى الطبقة العليا على أدوات إدارة المشـــروع وأدوات التحليل وأدوات التصميم، فى حين تحتوى الطبقة السفلى على أدوات البرمجة والاختبار والتوثيق والتدريب والتثبيت والصيانة (Hoffer et. Al, 2002).

تعتمد أدوات هندسة البرمجيات المدعمة بالحاسب على استخدام تقنية المستودع المركـــزى Repository لتخزين جميع المعلومات عن كل عملية من عمليات دورة حياة تطويـــر النظم. ويعتبر هذا المســـتودع بمنزلة قاعدة بيانـــات مركزية تحتوى على كل المعلومات الخاصة بمخططات وصف النظام مثل مخططات تدفق البيانات ومخطط العلاقـــة بيـــن الكائنات ومنطـــق العمليات وتعريف هيـــاكل البيانـــات، بالإضافة إلى

خصائص نماذج وتقارير النظام وخصائص واجهات استخدام النظام. ويتم استخدام هذا المســتودع فى إمداد الأدوات المختلفة بالمعلومــات التفصيلية التى تحتاج إليها لإنتاج المخرجات المطلوبة منها بدءاً من خطة المشــروع إلــى إنتاج الأكواد واختبار وصيانة البرمجيات المنتجة. ويتم أيضاً تســجيل جميــع التعديلات التى تحدث فى أثناء عمليات التطوير فى هذا المستودع. ويوضح شكل (٣-١٧) مكونات معظم أدوات هندسة البرمجيات المدعمة بالحاسب (Sommerville, 2004).

أدوات الطبقة العليا لـ CASE Tools	**أدوات إدارة المشروع:** تحديد الأهداف واختيار المشروع. تخطيط المشروع. مراقبة تنفيذ المهام والتحكم فيها.
	أدوات التحليل: تحديد متطلبات المشروع. نمذجة المتطلبات.
أدوات الطبقة السفلى لـ CASE Tools	**أدوات التصميم:** التصميم المنطقى. التصميم المادى.
	أدوات التطبيق: البرمجة. الاختبار. التوثيق. التدريب. التثبيت. الصيانة.

شكل (٣-١٦) طبقات أدوات هندسة البرمجيات المدعمة بالحاسب



شكل (٣-١٧) مكونات أدوات هندسة البرمجيات المدعمة بالحاسب

تحتوى معظم أدوات هندسة البرمجيات على المكونات التالية:

- محرر المخططات Diagram Editor:

يستخدم لإنشاء النماذج التخطيطية مثل نماذج تدفق البيانات ونماذج العلاقة بين الكائنات ونماذج حالات الاستخدام ومخطط الأصناف وغيرها من المخططات، وذلك بشكل قياسى ويتم تخزينها فى مستودع النظام Repository.

- أدوات التحليل والتصميم Analysis & Design Tools:

تقوم بعملية التدقيق للتأكد من صحة واكتمال وتناسق مواصفات نماذج التحليل والتصميم للنظام واستخراج الأخطاء لتصحيحها.

- لغة استفسار Query Language:

تســمح للمشاركين فى تطوير النظام بالاستفسار والبحث عن أى جزء من مكونات النظام فى مستودع النظام.

- قاموس البيانات Data Dictionary:

يحتوى على معلومات تفصيلية عن جميع مكونات النظام وتكون مخزنة فى مستودع النظام Repository، ويقوم أيضاً بتوليد توثيق النظام System Documentations.

- أدوات توليد النماذج والتقارير Forms & Reports Generation Tools:

تقـــوم بتصميم وجهـــات اســـتخدام النظـــام User Interface وتصميـــم التقارير المطلوبة.

- وسائل توريد واستيراد Import/Export Facilities:

تسمح بتبادل المعلومات بين المستودع وأدوات التطوير الأخرى.

- مولدات الأكواد Code Generators:

تقوم بإنتاج أكواد البرامج وقواعد البيانات آلياً وذلك باستخدام مواصفات التصميم المخزنة فى مستودع النظام.

- إدارة المشروع Managing the Project:

هـــى أدوات تقوم بتنفيذ أنشـــطة إدارة المشـــروع مثل تحديد الأهـــداف والمهام وجدولتها وتنظيم الموارد ومتابعة تنفيذ المهام وتعديل الخطة وغيرها.

تعتبر أداة Oracle Designer المقدمة من شـــركة ORACLE العالمية واحدة من أشهر أدوات هندسة البرمجيات المستخدمة حالياً فى تطوير مشاريع البرمجيات.

الفصل الرابع
إدارة مشاريع البرمجيات
Software Project Management

تعتبر إدارة المشروع من الأمور المهمة جداً عند البدء فى تنفيذ مشروع جديد أو تطوير مشروع قائم. تركز عملية إدارة مشاريع البرمجيات على التأكد من الاستخدام الجيد للموارد المخصصة لإنتاج برمجيات تفى بتطلعات المستفيد، وأن يتم تسليم المشروع فى الوقت المحدد وفى حدود الميزانية المتفق عليها. وعلى هذا فيمكننا تعريف إدارة المشروع بأنها عملية مخططة تقوم على تطبيق مجموعة من المعارف Knowledge والمهارات Skills والأدوات Tools والأساليب Techniques على نشاطات المشروع لتحقيق متطلبات وتوقعات المساهمين فى المشروع وذلك فى حدود تقيد زمنى ومالى.

مراحل إدارة مشاريع البرمجيات The Phases of Software Project Management:

تمر عملية إدارة مشروع البرمجيات بأربع مراحل رئيسية هى (شكل (٤-١)):

١- التخطيط Planning.

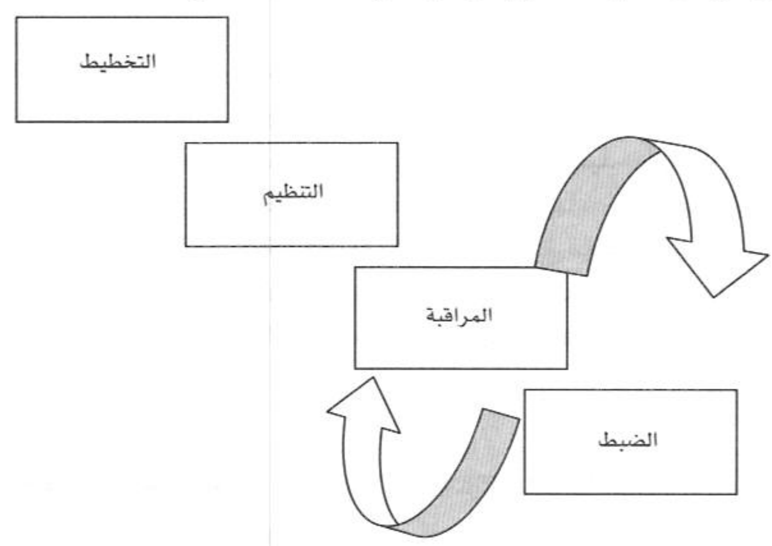
٢- التنظيم Organization.

٣- المراقبة Monitoring.

٤- الضبط Adjusting.

تبدأ دائماً عملية إدارة المشروع بتحديد أهداف مشروع الأعمال ويقوم بها مدير المشروع بالتعاون مع العميل أو صاحب المشروع. وتعتمد هذه العملية على عدة عناصر أساسية هى جودة البرمجيات المطلوبة والفترة الزمنية المتاحة لتنفيذ المشروع والميزانية المقدرة أو المتاحة حالياً لتنفيذه. وبعد ذلك يقوم مدير المشروع بعمليات تخطيط مهام المشروع وتنظيمها ومراقبة تنفيذها والتحكم وضبط عمليات تنفيذ مهام المشروع. يتم ذلك بأسلوب يماثل طريقة الشلال فى عملية تطوير البرمجيات، مع ملاحظة أنه ليس من الضرورى أن تكون جميع أنشطة إدارة المشاريع تتابعية،

ولكن تتداخل أحياناً بعض الأنشطة فى مرحلة معينة، أو حتى تتداخل بعض الأنشطة فى مراحــل مختلفة، مثلما يحدث بيــن مرحلتى المراقبة والضبــط، فنتيجة لعملية المراقبــة يمكن أن يقوم مدير المشــروع بإجراء تعديل أو ضبط فى خطة المشــروع الأولية أو فى عملية التنظيم أو كليهما (السماك، ٢٠٠٥).



شكل (٤-١) عملية إدارة مشاريع البرمجيات

التخطيط Planning:

تركــز مرحلــة التخطيط على إعداد خطة المشــروع التى تحتــوى على مجموعة الأنشطة التالية:

- وصف للمنتج المطلوب (برمجيات) وخصائصه.
- تحديد مهام المشروع.
- جدولة مهام المشروع.
- تحديد الموارد المطلوبة لإنجاز المشروع.
- تقدير ميزانية المشروع.
- تحديد المخاطر التى يمكن أن تواجه المشروع.

التنظيم Organization:

تقوم عملية تنظيم المشـــروع، بناء على خطة المشـــروع، بتنظيـــم الخدمات التى تســـاهم فى دعم عملية تنفيذ خطة المشـــروع بنجاح، وذلك من خلال تنفيذ مجموعة الأنشطة التالية:

- اكتساب الأفراد ذوى الخبرات المطلوبة للمشروع.
- تحديد المنهجيات والطرق التى سوف تستخدم فى تنفيذ عمليات المشروع.
- توفير مجموعة الأدوات التى تدعم تنفيذ عمليات المشروع.
- تحديد المعايير والقياسات التى سوف تستخدم لتقييم مخرجات المشروع.

ومن المهام الرئيســـية لعملية تنظيم المشـــروع هى عملية التأكـــد من أن الأفراد القائمين على العمل بالمشروع مؤهلين ومدربين جيداً على استخدام الأدوات المطلوبة، والمنهجيات المختارة، وتطبيق المعايير والقياسات المحددة.

المراقبة Monitoring:

تركز عملية مراقبة مشروع البرمجيات على الأنشطة التالية:

- تجميع البيانات المطلوب قياسها بشكل منتظم.
- تحليل البيانات ومعايرتها.
- تمثيل وعرض البيانات فى شكل تقارير دورية.
- إصدار توصيات ومقترحات بناء على البيانات التى تم تحليلها.

وبذلك فإن عملية المراقبة تهدف أساساً إلى التأكد من مطابقة العمليات الجارى تنفيذها لما هو محدد فى خطة المشـــروع وإمكانية إجراء التعديلات المناســـبة فى الوقت المناسب.

الضبط Adjusting:

من النادر جداً أن يقوم مدير المشـــروع بوضع خطة محكمة ويؤســـس تنظيماً لها بـــدون حدوث أى تعديلات عليها فـــى أثناء مرحلة التنفيذ. هناك تداخل مقصود بين مرحلتـــى المراقبة والضبط يؤدى إلى إمكانية إجراء التعديلات المناســـبة فى مرحلة

مبكرة فى أثناء تنفيذ المشــروع مما يقلل من المخاطر التى تؤثر فى نجاح المشروع. أحياناً تكون التعديلات طفيفة ولا تؤثر فى ســير المشــروع حسب الخطة الموضوعة مســبقاً، وأحياناً أخرى تكون التعديلات كبيرة وجوهرية، نتيجة أخطاء جسيمة حدثت فى أثناء عملية تحديد المتطلبات أو فى أثناء عملية تحديد وتقدير الموارد المطلوبة أو نتيجــة حدث طارئ مثل فقد أحد الأعضاء المهرة بفريق العمل، كل ذلك يؤدى إلى إعادة النظر فى خطة المشروع وتعديلها بشكلٍ كبير سواء فى محتواها أو فى الخطة الزمنية أو فى الموارد أو جميعها. تحدث دائماً عمليات التعديل والضبط تحت ضغط وظروف غير مريحة، ويتطلب ذلك أن يتحلى مدير المشروع بالثبات والجرأة؛ ليتمكن من اتخاذ القرارات المناسبة لتصحيح الأوضاع فى الوقت المناسب بأقل الخسائر مع تجنب التأثير فى نجاح المشروع.

٤-١ تطوير خطة المشروع Project Plan Development:

تعتبــر عملية تخطيط المشــروع بمنزلة الرأس لباقى جســم المشــروع. فعملية التخطيط تقوم بتحديد الأفعال (المهام)، وتوقيت تنفيذها، ومن يقوم بتأديتها، ووضع بدائــل احتياطية لها، وجدولة المهــام للرجوع إليها فى أثناء عمليــة متابعة التنفيذ، واتخــاذ أفعال تصحيحية على الخطة لضبط المشــروع. ولذلــك فإنه يمكننا القول بأن أنشــطة تخطيط المشــروع تقوم بدور ضابط الإيقاع أو المايسترو لباقى مراحل المشــروع. وعلى هذا فإنه إذا لم يتم تخصيص وقت كافٍ لمرحلة التخطيط أو وضع خطة متعجلة وغير دقيقة للمشروع، فسوف يؤدى ذلك إلىً فشل المشروع فى الغالب. ومــن جانب آخر، فإن وضع خطة محكمة ودقيقة للمشــروع لا يضمن بالضرورة عدم حدوث مشــكلات فى أثناء التنفيــذ، ولكنها بالقطع تزيد من فرص حل المشــكلات والتغلب عليها، مما يقلل أو يحد من احتمالية فشل المشروع.

يقــوم مدير المشــروع بوضع خطة المشــروع بالتعاون مع أغلب المشــاركين فى المشــروع (الممول، العميل، المستخدمين، فريق التطوير، ......) بقدر الإمكان. وبعد إعــداد الخطة، يجب أن يقوم جميع المشــاركين فى المشــروع بمراجعتها والموافقة عليها. ومن الضرورى أيضاً أن يكون مدير المشروع على دراية تامة بمبررات الاحتياج للمشــروع (مثل تحســين درجة رضا العميل، وتحسين الأداء، وزيادة الإنتاجية، وتقليل التكلفة، ..) حتى يمكنه ترجمة ذلك إلى أهداف واضحة يمكن قياســها وتضمينها فى خطة المشروع (Sommerville, 2004).

تحتــاج عملية تطوير خطة مشـــروع البرمجيات إلى إعــداد مجموعة من الخطط التفصيلية مثل:

- خطة الجدولة Schedule Plan.
- خطة الميزانية Budget Plan.
- خطة الجودة Quality Plan.
- خطة الاختبار Test Plan.
- خطة تنمية الأفراد People Development Plan.
- خطة إدارة المخاطر Risk Management Plan.
- خطة إدارة التشكيل Configuration Management Plan.

يوضح شـكل (٤-٢) مخطط النشــاط الذى يمثل أنشــطة تطوير خطة المشروع. تبدأ الخطة بنشــاط تحديد مجال وأهداف المشروع، وبعد ذلك يقوم مدير المشروع بتجزئة أعمال المشــروع، ومن ثم تحديد المهام Tasks والمســتلمات Deliverables والمعالــم Milestones للمشــروع. وبعد ذلك يقوم مدير المشــروع بتحديد الموارد المطلوبة لتنفيذ مهام المشروع التى تتضمن الموارد البشرية والموارد المادية. وبعد ذلك يقوم بتحديد مجموعة المخاطر التى يمكن أن تواجه المشروع فى أثناء التنفيذ، وكذلك تحديــد معايير الجودة المطلوبة لمخرجات المشــروع. وبناءً على ذلك يقوم مدير المشروع بإعداد الخطط التالية:

- خطة تأسيس وتنمية الموارد.
- خطة الاتصال بين فرق العمل.
- خطة إدارة المخاطر.

وبعــد ذلك يقوم بإعداد جدول مهام المشــروع وتخصيص الموارد لكل مهمة ويتم اســتخدام معلومــات الجدولة فــى تقدير ميزانية المشــروع. يقوم مدير المشــروع باســتخدام جميع المعلومات السابقة فى إعداد خطة المشروع الأساسية Base Line Plan، ومن ثم يقوم بمراقبة التقدم فى تنفيذ مهام المشروع واتخاذ الأفعال التصحيحية المناسبة لضبط عمليات التنفيذ والتحكم فى مخرجات المشروع لتحقيق أهداف خطة المشروع. وفى أثناء عمليات التنفيذ يواجه المشروع غالباً بعض المخاطر والتحديات التى تعوق إنجاز المهام حسب الخطة الموضوعة. وإذا كانت قيود الميزانية والجدول الزمنى للمشــروع غير قابلة للتعديل، فيضطر مدير المشروع إلى إجراء تعديلات فى أهداف أو قيود المشــروع الأخرى مثل قيود الجودة. أما إذا كانت قيود الميزانية أو

الجدولة غير ثابتة، فإنه يقوم بإعادة جدولة المهام والموارد بالمشروع حسب الاحتياج لتحقيق الأهداف بمواصفات الجودة المطلوبة بالخطة. سنقوم الآن بعرض موجز لأهم أنشطة تطوير خطة مشروع البرمجيات.

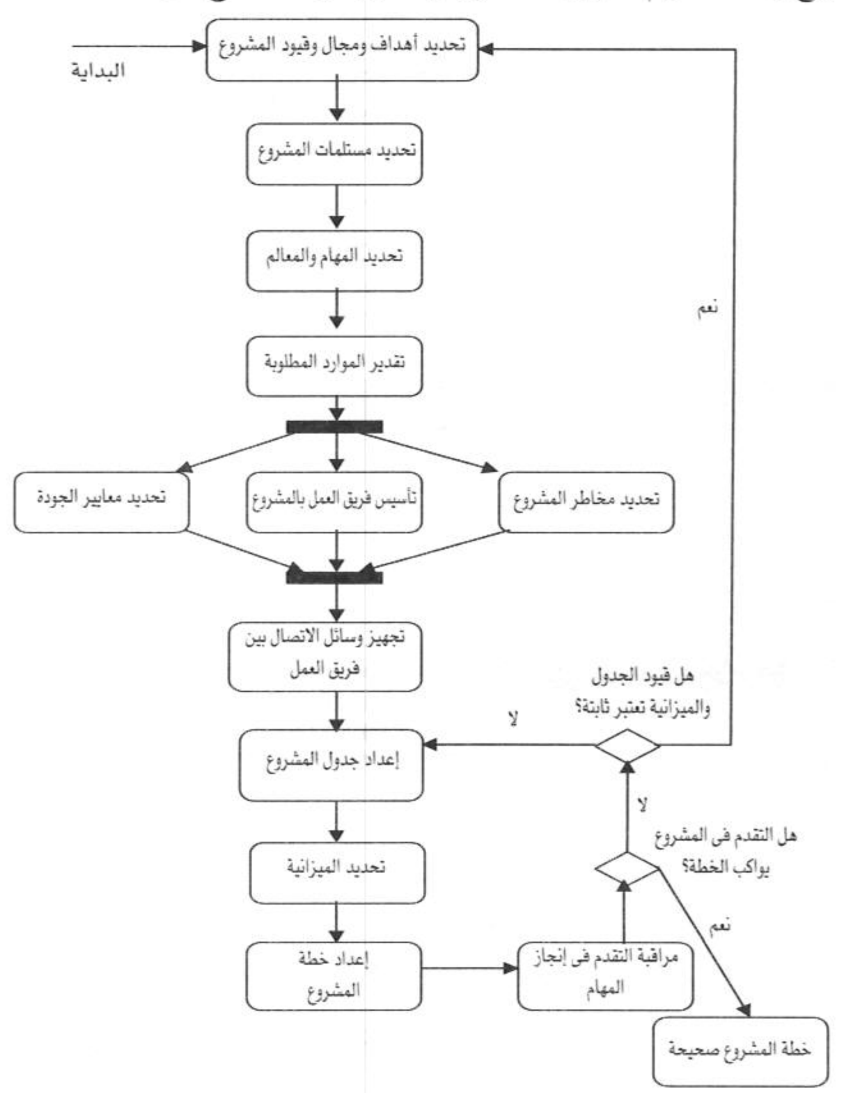
٤-٢ جدولة المشروع Project Scheduling:

تعتبر عملية جدولة مهام المشروع ومراجعتها من الأنشطة المهمة لمدير المشروع. تتم هذه العملية بعد إتمام عملية تحديد قائمة مهام المشروع. ولأن كل مهمة أو نشاط فى المشروع له وقت محدد لبدايته وفترة زمنية محددة لتنفيذه ومن ثم له نهاية محددة، فإن مدير المشروع يقوم بعملية جدولة مهام المشروع بناء على هذه المعلومات بالإضافة إلى مجموعة من القيود التى تنظم عملية الجدولة مثل عملية تخصيص الموارد المطلوبة لكل مهمة. ومن الطبيعى أن تتعرض عملية تنفيذ المهام لبعض المشكلات التى تتطلب تدخل مدير المشروع لحلها وإعادة جدولة المهام حسب الظروف الطارئة حتى لا يؤثر التأخير فى إنجاز المهمة فى باقى أنشطة المشروع. يقوم مدير المشروع بالاستعانة ببعض الأدوات فى أثناء عمليات إدارة المشروع مثل برنامج Microsoft Project الذى يساعده فى جدولة المهام وتوزيع الموارد وإعادة الجدولة آلياً حسب المعطيات الجديدة عند حدوث تغييرات. تساعده هذه الأدوات أيضاً فى متابعة إنجاز المهام ومراقبة المهام الحرجة بدقة باستخدام مخطط المسار الحرج Critical Path Diagram الذى يحتوى على مجموعة المهام التى يجب تنفيذها فى الوقت المحدد لها فى الجدول لضمان انتهاء المشروع فى الوقت المقدر له فى خطة المشروع.

٤-٢-١ المهام والمستلمات Tasks and Deliverables:

كما ذكرنا سابقاً فإن عملية جدولة المشروع تعتمد على تحديد مهام وأنشطة المشروع. يقوم مدير المشروع - بمساعدة فريق العمل - بتنظيم مهام المشروع من خلال عدة مستويات هرمية تبدأ بالمهام العامة فى المستوى الأعلى ثم يتم تفصيلها فى المستويات التالية إلى أسفل فى شكل مهام فرعية. تعرف هذه العملية بعملية «تجزئة العمل Work Breakdown". يتم تحديد الفترة الزمنية المناسبة لإنجاز كل مهمة وهى تراوح بين عدة ساعات إلى عدة أسابيع حسب حجمها. ومن الأفضل ألا تزيد الفترة المخصصة للمهمة على أسبوعين. وبعد ذلك يبدأ مدير المشروع فى إعداد الجدول الزمنى للمشروع بدءاً من المهمة الأولى فى الترتيب ثم المهام الأخرى التى تليها حتى نصل لنهاية المشروع. وفى الحالات التى تتطلب أن يتم الانتهاء من المشروع فى وقت محدد، كأن يكون مرتبطاً بموسم معين يجب أن يكون المشروع

جاهزاً للتشغيل فيه، فإنه يمكن لمدير المشروع أن يبدأ بهذا الوقت ويخصصه لنهاية النشـــاط الأخير فى المشـــروع ويســـتمر فى جدولة باقى المهام فى الاتجاه العكسى حتـــى يصل إلى المهمة الأولى فى المشـــروع، من ثم يتم تحديد الوقت الملائم لبداية المشروع وكذلك حجم الموارد المطلوبة إذا كان الوقت المتاح ضيقاً.



شكل (٤-٢) مخطط أنشطة تخطيط المشروع

يتضمن المشروع بعض المهام التى تؤدى إلى حدث بارز أو تسليم مخرجات جوهرية وهى تعتبر معالم أو علامات فارقة Milestones فى المشـــروع. من الممكن أن تكون هذه المعالم دليلاً على نهاية عملية أو مرحلة أو المشـــروع نفسه. هذا بالإضافة إلى أن كل مهمة من مهام المشـــروع يمكـــن أن تؤدى إلى إنتاج مســـتلمات Deliverables محددة من مستلمات المشروع. أحياناً يتم تمثيل المعالم والمستلمات فى شكل مهام ليس لها فترة زمنية (duration = صفر). يوضح شكل (٤-٣) مثالاً لاستخدام برنامج Microsoft Project لإنشاء قائمة مهام أحد المشاريع. نلاحظ فى الشكل أن القائمة تركـــز على عرض المهام فى شـــكل هرمى وتوضيح الفترة الزمنيـــة المخصصة لكل مهمة. وقد تمت تســـمية المهام فى المســـتوى الأعلى باسم "Summary Task" ويتم حســـاب الفترة الزمنية لها آلياً وهى تســـاوى مجموع الفترات الزمنية للمهام الفرعية لها. ونلاحظ أيضاً وجود مهام روتينية Recurring Tasks تحدث بشـــكل دورى مثل اجتماع لمراجعة الخطة أسبوعيا لمدة ساعتين.

	Task Name	Duration	Start
1	تحليل النظام	5 days	Sat 12/05/07
2	تجميع المعلومات	5 days	Sat 12/05/07
3	نمذجة العمليات	5 days	Sat 12/05/07
4	نمذجة البيانات	3 days	Sat 12/05/07
5	قاموس البيانات	0 days	Sun 13/05/07
6	تصميم النظام	4 days	Sat 12/05/07
7	تصميم العمليات	4 days	Sat 12/05/07
8	تصميم قواعد البيانات	3 days	Sat 12/05/07
9	تصميم الواجهات	3 days	Sat 12/05/07
10	[illegible]	10 days	Sat 12/05/07
11	برمجة التطبيقات	10 days	Sat 12/05/07
12	إختبار التطبيقات	10 days	Sat 12/05/07
13	إجتماعات دورية	#########	Sun 13/05/07
14	مراجعة الخطة ١	2 hrs	Sun 13/05/07
15	مراجعة الخطة ٢	2 hrs	Sun 20/05/07
16	مراجعة الخطة ٣	2 hrs	Sun 27/05/07
17	مراجعة الخطة ٤	2 hrs	Sun 03/06/07

شكل (٤-٣) قائمة مهام المشروع

٤-٢-٢ جدولة المهام Task Scheduling:

تعتمد جدولة المشـــروع على الترتيب الزمنى لأنشـــطة المشروع وعلى الاعتمادية بينها. ولهذا يجب تحديد الاعتمادية بين مهام المشـــروع قبل عملية الجدولة. غالباً

ما توجد الاعتمادية بين نشاطين أو أكثر نتيجة لطبيعة تنظيم إجراءات العمل أو نتيجة لاشـتراك أكثر من مهمة فى اسـتخدام الموارد نفسها خصوصاً البشرية. أما المهام التى لا تعتمد على بعضها فيمكن لمدير المشـروع أن يجدولها فى الوقت نفسـه (أى تعمل عل التوازى) أو حسب ظروف الموارد المطلوبة. تقدم أداة Microsoft Project أربعة أنواع من الاعتمادية بين المهام هى:

١- نهاية - إلى - بداية Finish-to-Start:

لا يمكن أن تبدأ المهمة التابعة successor إلا بعد انتهاء المهمة السابقة لها predecessor.

٢- بداية - إلى - بداية Start-to-Start:

لا يمكن أن تبدأ المهمة التابعة إلا بعد بداية المهمة السابقة لها.

٣- نهاية - إلى - نهاية Finish-to-Finish:

لا يمكن أن تنتهى المهمة التابعة إلا بعد انتهاء المهمة السابقة لها.

٤- بداية - إلى - نهاية Start-to-Finish:

لا يمكن أن تنتهى المهمة التابعة إلا بعد بداية المهمة السابقة لها.

وبعد تحديد تلك التبعيات بين المهام، فإن أى تعديل يحدث فى بداية أى مهمة أو فى الفترة الزمنية لها سوف يؤثر آلياً فى جدولة المهام التالية لها.

هناك نوعان آخران للتبعية بين المهام هما:

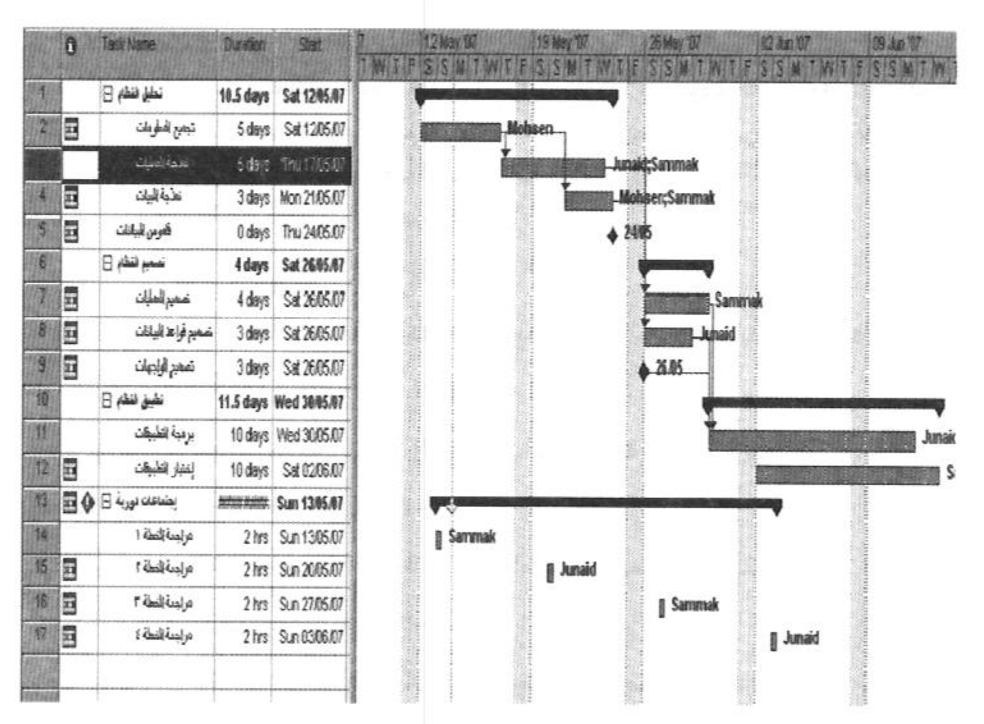
- تبعيات التأخير Delay Dependencies:

تحدد فترة زمنية محددة بعد نهاية المهمة السابقة حتى تبدأ المهمة التالية لها.

- تبعيات متداخلة Overlap Dependencies:

يحدث فيها تداخل بين المهام، إذ يمكن أن تبدأ مهمة قبل نهاية المهمة السـابقة لها بفترة معينة.

الآن وبعـد تحديد قائمـة المهام وتحديـد التبعية بينها، يتم إنشـاء جدول مهام المشروع باستخدام أسلوب خريطة جانت GANTT Chart. يوضح شكل (٤-٤) مثالا لاسـتخدام أداة Microsoft Project فى إنشـاء الجدول الزمنى لمهام أحد مشـاريع البرمجيات.



	Task Name	Duration	Start
1	تحليل النظام	10.5 days	Sat 12/05/07
2	تجميع المعلومات	5 days	Sat 12/05/07
3	نمذجة العمليات	5 days	Thu 17/05/07
4	نمذجة البيانات	3 days	Mon 21/05/07
5	قاموس البيانات	0 days	Thu 24/05/07
6	تصميم النظام	4 days	Sat 26/05/07
7	تصميم العمليات	4 days	Sat 26/05/07
8	تصميم قواعد البيانات	3 days	Sat 26/05/07
9	تصميم الواجهات	3 days	Sat 26/05/07
10	تطبيق النظام	11.5 days	Wed 30/05/07
11	برمجة التطبيقات	10 days	Wed 30/05/07
12	إختبار التطبيقات	10 days	Sat 02/06/07
13	إجتماعات دورية	########	Sun 13/05/07
14	مراجعة الخطة ١	2 hrs	Sun 13/05/07
15	مراجعة الخطة ٢	2 hrs	Sun 20/05/07
16	مراجعة الخطة ٣	2 hrs	Sun 27/05/07
17	مراجعة الخطة ٤	2 hrs	Sun 03/06/07

شكل (٤-٤) جدولة مهام أحد المشاريع باستخدام خريطة جانت

٤-٢-٣ تقويمات الموارد Resources Calendars:

يتم تصنيف الموارد المطلوبة لإنجاز كل مهمة فى المشروع إلى نوعين:

- موارد بشرية:

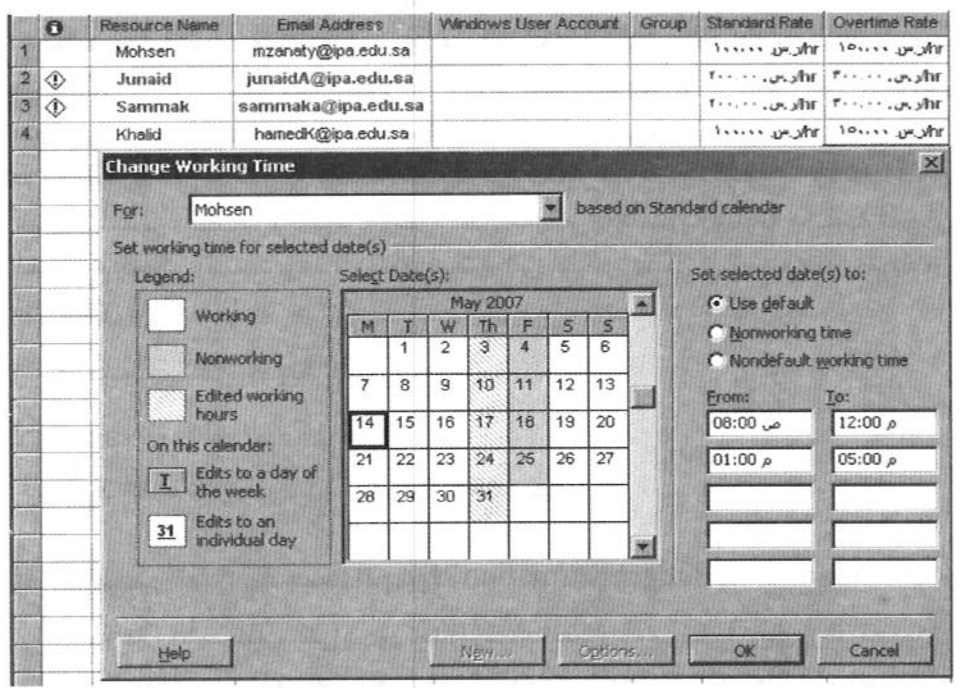
تصف مجموعة الأفراد القائمين على إنجاز المهمة.

- موارد مادية:

تصف مجموعة الأجهزة والبرمجيات والمواد المطلوب استخدامها لإنجاز المهمة.

يمكن أن تحتاج المهمة إلى أكثر من مورد سواء من كلا الصنفين أو أحدهما. وقبل عملية تخصيص الموارد على المهام، يجب تحديد الموارد المطلوبة ونوعها وكميتها والوقت المطلوب توفيرها فيه وفترة استخدامها. وبعد انتهاء عملية تخصيص الموارد

على المهام، يقوم مدير المشـــروع بمتابعة وتقييم الموارد المستخدمة. فيقوم بتقييم حجم الإنجاز للموارد البشرية وكذلك معدل استخدام الموارد المادية. وبناءً على ذلك يمكن لمدير المشـــروع أن يعيد تخصيص الموارد على المهام حسب معدل الإنجاز أو الاســـتخدام، فيمكنه الاستفادة من الموارد القليلة الاستخدام فى إحدى المهام بإعادة تخصيصها لمهام أخرى تحتاج إليها. وكذلك يقوم مدير المشروع بتقييم تكلفة الموارد المســـتخدمة. تدعم الكثير من أدوات إدارة المشـــروع إمكانية التعديل فى معلومات الموارد المستخدمة فى المشروع من خلال استخدام تقويم يطلق عليه «تقويم الموارد Resource Calendar" الذى يمكن مدير المشروع من إدخال أوقات عمل أو استخدام الموارد وكذلك مواعيد التوقف عن العمل كما هو موضح بشكل (٤-٥).



شكل (٤-٥) تقويم الموارد

٤-٢-٤ جدولة المجهود Effort Scheduling:

تعتمد جدولة المشـــروع على تخصيـــص الموارد على مهام المشـــروع هى تعتمد بدورها على فترة اســـتخدام الموارد. فمثـــلاً إذا كان أحد الأفراد يعمل لفترة نصف

الوقت وبدون قيود فى تقويم الموارد، فإن هذا الفرد يكون متاحاً لفترة (٤) ساعات فى اليوم. وإذا كان حجم العمل المكلف به الفرد يزيد على هذه المدة فيعتبر الفرد مكلفاً بعمل إضافى، مما يؤدى إلى تقليل فترة إنجاز المهمة ومن ثم يؤثر إيجابياً فى جدول المشروع. وعلى هذا فإن إجمالى المجهود المطلوب لإنجاز المهمة لم يتغير ولكن فترة الإنجاز هى التى تغيرت. وفى المقابل إذا كانت فترة إنجاز المهمة غير قابلة للتغيير، فيمكن لمدير المشروع أن يخصص الموارد المطلوبة لبذل المجهود المطلوب لإنجاز المهمة فى الوقت المحدد.

٤-٣ تقدير ميزانية المشروع Project Budget Estimation:

تعتبر عملية تقدير ميزانية المشروع من الأنشطة المهمة والحرجة لمدير المشروع. هناك العديد من الأساليب المقترحة لتقدير التكاليف بعضها لا يعتمد على أساس علمى، والبعض الآخر يعتمد على خبرة القائمين على عملية التقدير استناداً إلى خبرة هؤلاء الأفراد فى مشاريع سابقة مشابهة فى تقدير ميزانية المشروع. وهناك طرق أخرى تسمى «التقدير بالقياس» وتقوم على القياس بمشاريع سابقة مشابهة لمجال المشروع الحالى نفسه. وهناك طرق تستند إلى أساس علمى تستخدم منها طريقتان على نطاق واسع هما:

١- تقدير مبنى على جدولة المشروع.

٢- تقدير مبنى على الخوارزميات.

وسوف نقوم الآن بعرضهما بشىء من التفصيل.

٤-٣-١ تقدير الميزانية حسب الجدولة Schedule-driven Budget Estimation:

يعتمد هذا الأسلوب على وجود جدول لمهام المشروع مسبقاً وأن يكون قد تم توزيع الموارد المطلوبة على المهام. وتعرف هذه العملية بأنها تمثل الجدولة الأساسية أو المبدئية للمشروع وذلك لإمكانية التعديل فيها لاحقاً حسب التغيرات التى تطرأ على المشروع، مما يؤدى إلى وجود أكثر من جدول للمشروع. وبعد ذلك يتم تقدير تكلفة كل مهمة فى الجدول بدقة ويتم تجميعها ثم تضاف إليها التكاليف الثابتة للمشروع Fixed Costs. يتم إدخال تلك المعلومات من خلال استخدام إحدى أدوات إدارة المشاريع مثل Microsoft Project لحساب الميزانية المقترحة للمشروع. يجب الأخذ فى الاعتبار

أن تلـــك التقديرات تكون معرضة للتغيير فى أثناء تنفيذ المشـــروع نظراً لإمكانية تغير تكلفـــة الموارد المطلوبة عند طلبها عن التكلفة المقدرة أثناء إعداد الخطة بشـــكل كبير نظراً لظروف غير متوقعة مثل الحروب أو الأزمات. يتم تقدير تكلفة الموارد البشـــرية عن طريق حســـاب معدل الأجر اليومى أو الأسبوعى أو الشهرى أو خلال فترة محددة للأفراد المشاركين فى عمليات التطوير، وكذلك حساب معدل الأجر الإضافى المحتسب لســـاعات العمل الإضافية التى يكلف بها بعض الأفـــراد لإنجاز مهامهم فى وقت أقل مـــن المقدر فى الخطة. يتم تجميع كل هذه التكاليف ويضـــاف إليها التكاليف الثابتة لكل مهمة. يوضح شـــكل (٤-٦) مثالاً لاستخدام أداة Microsoft Project فى حساب التكلفة الكلية لكل مهمة لأحد المشاريع.

	ℹ	Task Name	Duration	Start	Finish	Predecessors	Resource Names	Fixed Cost	Baseline Cost	Total Cost
1		⊟ تحليل النظام	**10.5 days**	**Sat 12/05/07**	**Wed 23/05/07**			ر.س. ٥٠٠٫٠٠	ر.س. ٢٧،٢٠٠٫٠٠	27700
2	✓	تجميع المعلومات	5 days	Sat 12/05/07	Wed 16/05/07		Mohsen	ر.س ٠٫٠٠	ر.س ٤،٠٠٠٫٠٠	4000
3	✓	نمذجة العمليات	5 days	Thu 17/05/07	Wed 23/05/07	٢	Junaid;Sammak	ر.س ٠٫٠٠	ر.س ١٦،٠٠٠٫٠٠	16000
4		نمذجة البيانات	3 days	Mon 21/05/07	Wed 23/05/07	٢FS+٢ days	Mohsen;Sammak	ر.س ٠٫٠٠	ر.س ٧،٢٠٠٫٠٠	7200
5		قاموس البيانات	0 days	Thu 24/05/07	Thu 24/05/07		Junaid;Mohsen	ر.س ٥٠٠٫٠٠	ر.س ٠٫٠٠	500
6		⊟ تصميم النظام	**4 days**	**Sat 26/05/07**	**Tue 29/05/07**			ر.س. ٦٠٠٫٠٠	ر.س. ١٣،٦٠٠٫٠٠	14200
7		تصميم العمليات	4 days	Sat 26/05/07	Tue 29/05/07	٣	Sammak	ر.س ٠٫٠٠	ر.س ٦،٤٠٠٫٠٠	6400
8		تصميم قواعد البيانات	3 days	Sat 26/05/07	Mon 28/05/07	٤	Junaid	ر.س ٠٫٠٠	ر.س ٤،٨٠٠٫٠٠	4800
9		تصميم الواجهات	3 days	Sat 26/05/07	Mon 28/05/07		Mohsen	ر.س ٦٠٠٫٠٠	ر.س ٢،٤٠٠٫٠٠	3000
10		⊟ تطبيق النظام	**15.5 days**	**Sat 26/05/07**	**Wed 13/06/07**			ر.س. ١٠٠٠٫٠٠	ر.س. ٥١،٦٠٠٫٠٠	52600
11		برمجة التطبيقات	11.5 days	Sat 26/05/07	Sat 09/06/07	٧،٨،٩	Junaid;Khalid	ر.س ٠٫٠٠	ر.س ٢٧،٦٠٠٫٠٠	27600
12		إختبار التطبيقات	10 days	Sat 02/06/07	Wed 13/06/07		Sammak;Mohsen	ر.س ٠٫٠٠	ر.س ٢٤،٠٠٠٫٠٠	24000
13		⊟ إجتماعات دورية	**#########**	**Sun 13/05/07**	**Sun 03/06/07**			ر.س. ١،٢٠٠٫٠٠	ر.س. ١،٦٠٠٫٠٠	2800
14	✓	مراجعة الخطة ١	2 hrs	Sun 13/05/07	Sun 13/05/07		Sammak	ر.س ٠٫٠٠	ر.س ٤٠٠٫٠٠	400
15	✓	مراجعة الخطة ٢	2 hrs	Sun 20/05/07	Sun 20/05/07		Junaid	ر.س ٠٫٠٠	ر.س ٤٠٠٫٠٠	400
16	✓	مراجعة الخطة ٣	2 hrs	Sun 27/05/07	Sun 27/05/07		Sammak	ر.س ٠٫٠٠	ر.س ٤٠٠٫٠٠	400
17		مراجعة الخطة ٤	2 hrs	Sun 03/06/07	Sun 03/06/07		Junaid	ر.س ٠٫٠٠	ر.س ٤٠٠٫٠٠	400

شكل (٤-٦) حساب تكلفة المشروع بناءً على جدول المشروع

نلاحظ من الشكل ما يلى:

- يتم إضافة التكلفة الثابتة لمعالم Milestones ومستلمات Deliverables كل مهمة.
- يتم إضافة التكلفة الثابتة لمهام المراجعة Recurring Tasks.
- تعتبر التكلفة الكلية هى تكلفة الخطة الأساســية Baseline للمشــروع مضافاً إليها التكلفة الثابتة للمهام.

٤-٣-٢ تقدير الميزانية حسابياً Algorithmic Budget Estimation:

مع كثرة الاختلافات التى حدثت فى مشاريع كثيرة فى السابق بين التكلفة المقدرة والتكلفــة الفعلية للمشــروع، أصبح لــدى القائمين على إدارة المشــروع قناعة كبيرة بضرورة أخذ طرق أخرى فى الحســبان لتقدير تكلفة المشــروع، إذ لا يجب الاعتماد على طريقة واحدة فحسب. ولذلك فإنه من الأفضل الاستعانة بالطرق الرياضية فى تقديــر التكلفة للتحقق من دقة الطرق الأخرى. تعتمد الطرق الحســابية على بعض الصيغ الرياضية لتقدير تكلفة الموارد البشرية على وجه الخصوص من حيث حساب تكلفة المجهود المبذول وحجم المشــروع. يتم تمثيل حجم المشــروع من خلال عدة عناصر مثل:

أ- عدد أسطر الأكواد Lines of Code (LOC):

يصف عدد أســطر أكــواد البرامج المكتوبة بــدون الأخذ فى الاعتبار الأســطر التوضيحية Comment lines. وتحسب الأسطر دائماً بالآلاف وتأخذ الرمز KLOC فى الصيغ الرياضية.

ب - نقاط الوظيفة Function Points (FPs):

تعتبر أحد مقاييس حســاب حجم المشــروع ولكن فى مســتوى أعلى من مستوى عدد الأسطر. يتم حسابها بناءً على توصيف المنتج البرمجى من خلال وظائف أساسية، هى:

- المدخلات Inputs.
- المخرجات Outputs.
- ملفات البيانات Data Files.
- الاستفسارات Inquiries.
- الواجهات الخارجية External Interfaces.

يتم حساب عدد النقاط حسب تلك الوظائف مع الأخذ فى الاعتبار معاملات وزنية Weighting factors لضبط عملية حساب التكلفة حسب درجة تعقيد الوظيفة.

ج - نقاط الكائن (Ops) Object Points:

تمثل مقياســـاً آخر فى مستوى أعلى لحســـاب حجم المشروع. وهى أيضاً تعتمد على توصيف المنتج البرمجى من خلال مجموعة من الخصائص المقدمة مثل:

- المكونات البرمجية Software Components.
- التقارير Reports.
- واجهات المستخدم User Interfaces.

يتم حســـاب مجموع النقاط للكائنات مع الأخذ فى الاعتبار معاملات وزنية لضبط عملية حساب التكلفة حسب درجة تعقيد كل كائن.

يتم اســـتخدام هذه المعلومات عن حجم المشـــروع فى صياغة النموذج الرياضى المستخدم فى تقدير تكلفة المشروع ويأخذ الشكل العام:

$$Effort = C * Size^K$$

حيث تأخذ المعاملات C،K قيماً تجريبية مأخوذة من مشاريع سابقة. يمثل المعامل K درجة التعقيد بالمشروع وتأخذ القيم من ١ إلى ١,٥، ويمثل المعامل C درجة الصعوبة فى المشروع التى يتم تقييمها من خلال عدة خصائص مثل مهارات فريق العمل ودرجة إتاحة الموارد المطلوبة ودرجة المرونة المطلوبة للمنتج البرمجى وغيرها.

ســـنقدم الآن عرضاً لأحد أشـــهر الطرق الرياضية المســـتخدمة فى تقدير تكلفة المشروع وهى طريقة COCOMO.

طريقة COCOMO:

يهدف النموذج الرياضى Constructive Cost Model إلى حساب تكلفة المجهود المبـــذول لإنجـــاز مهام المشـــروع. لقـــد تم تقديم أول نســـخة منه فـــى عام ١٩٨١ باســـم COCOMO 81، وتم تطويرها فيما بعد لتظهر النســـخة المطورة عام ٢٠٠٠ باســـم COCOMO II وســـوف نركز الآن على النسخة المطورة للأســـباب التالية: (Pressman, 2001).

- تناسب أســـاليب التطوير الحديثة مثل أسلوب التكرار مع الزيادة وأسلوب البرمجة الموجهة بالكائنات Object-Oriented Programming.
- تناسب أيضاً استخدام أساليب النمذجة Prototyping واستخدام مولدات البرامج والتطوير باستخدام المكونات Component-Based Development.
- تلائم التطبيقات الكبيرة والمعقدة.

ومن السـمات المهمة لطريقة COCOMO II أنهـا تعتمد على مرحلة التطوير التى يتـم التقديـر خلالها، مما يـؤدى إلى دقة عمليـة التقدير للتكلفة. ولهـذا فإن طريقة COCOMO II تقدم ثلاثة نماذج مختلفة تقابل مراحل التطوير الأولى للمشروع هى:

١- نموذج تكوين التطبيق Application Composition Model:

يسـتخدم فى بداية عملية تحليل المتطلبات، ويسـتخدم نقـاط الكائن الموزونة Weighted Object Points (Ops) لتحديـد حجـم المهمـة المطلوب تقديرها. يتم حسـاب نقاط الكائن لكل من واجهات المسـتخدم وتقارير المنتج البرمجى ومكونات المنتج البرمجى. وبعد ذلك يتم تخفيض عدد النقاط حسـب نسبة الكائنات التى تم الحصول عليها باسـتخدام أسلوب إعادة الاستخدام Reuse Objects واحتساب نقاط الكائنات الجديدة فقط. يتم صياغة معادلة حساب نقاط الكائن (NOP) كما يلى:

$$NOP = OP * [(100 - \% Reuse)] / 100]$$

يستخدم النموذج المقترح عاملين أساسيين لتحديد مستوى إنتاجية عملية التطوير Development Productivity Level وهمـا عامل الخبـرة والمقدرة لفريق التطوير وعامل مكانة ومقدرة المؤسسـة. يوضح شـكل (٤-٧) المستويات الخمسة لإنتاجية عملية التطوير.

وبعد ذلك يمكننا حساب قيمة المجهود المطلوب من فريق العمل كما يلى:

$$Effort = NOP / Productivity$$

فمثلاً إذا كان عدد نقاط الكائن هى (٥٠٠) والإنتاجية هى (٤)، فإن قيمة المجهود المطلـوب هى (١٢٥) شخص/شـهر، وإذا كانت إنتاجية فريـق العمل هى (٥٠) فإن المجهود المطلوب يصبح (١٠) أشخاص/شهر فقط.

الإنتاجية NOP/month	خبرة ومقدرة المطورين ونضج ومقدرة المؤسسة
٤	قليلة جداً
٧	قليلة
١٣	عادية
٢٥	عالية
٥٠	عالية جداً

شكل (٤-٧) قياس الإنتاجية فى طريقة COCOMO II

٢- نموذج التصميم المبكر Early Design Model:

يستخدم بعد اكتمال عملية تحليل المتطلبات ويستخدم نقاط الوظيفة (FPs) لحساب حجم المهمة المطلوب تقديرها. تقدم طريقة COCOMO II جدولاً قياسياً لتحويل عدد نقاط الوظيفة إلى عدد أسطر الأكواد (KLOCs) الذى يستخدم بدوره فى تقدير قيمة المجهود المطلوب كما يلى:

$$\text{Effort} = C * \text{SizeK} * m + \text{autoeffort}$$

يتم تحديد قيم المعاملات المذكورة فى الصيغة بناءً على معلومات من (٨٣) مشروعاً سابقاً كما يلى:

- يأخذ المعامل C قيمة (٢,٥) فى مرحلة التصميم الأولية.
- يأخذ المعامل K قيماً تراوح بين (١,٠١) إلى (١,٢٦).
- يتم تحديد قيم المعامل m من خلال سبع خصائص هى:
- درجة الثقة فى المنتج البرمجى Reliability.
- درجة إعادة الاستخدام Reusability.
- درجة صعوبة بيئة التشغيل Platform.
- مقدرة الأشخاص Capability.
- خبرة الأشخاص Experience.
- الجدول Schedule.
- التسهيلات المقدمة Facilities.

تأخذ كل خاصية قيمة معينة، وإذا كانت الخاصية غير مؤثرة فإنها تأخذ القيمة (١). تكون قيمة المعامل m هى حاصل ضرب القيم السبع.

يعنى المعامل autoeffort بالمجهود الذى يقوم به المطورون فى أثناء عملية توليد الأكواد آلياً وكذلك عملية تكاملها مع الأكواد الأخرى التى ينتجها المطورون أنفسهم.

٣- نموذج ما بعد التصميم المعمارى Post-Architectural Model:

يستخدم بعد تحديد التصميم المعمارى للبرمجيات ويستخدم عدد سطر الأكواد (LOCs) لتحديد حجم المهمة المطلوب تقديرها. ويستخدم الصيغة الرياضية المذكورة نفسها فى النموذج السابق (نموذج التصميم المبكر) ولكن يختلف عنه فى تفاصيل

المعاملات. يستخدم هذا الأسلوب (١٧) خاصية موزعة على أربع مجموعات. تأخذ الخواص غير المؤثرة القيمة الأحادية، ويأخذ المعامل m حاصل ضرب قيم الخواص السبع عشرة. يوضح شكل (٤-٨) مجموعة الخواص التى تقود عملية تقدير التكلفة.

اسم الخاصية	كود الخاصية	تصنيف الخاصية
درجة الثقة المطلوبة للنظام	RELY	خصائص المنتج
درجة تعقيد وحدات النظام	CPLX	
درجة التوثيق المطلوب	DOCU	
حجم قواعد البيانات	DATA	
النسبة المئوية المطلوبة لإعادة استخدام المكونات	RUSE	
قيود وقت التنفيذ	TIME	خصائص الحاسب
فترة بقاء النظام الأساسى للتطوير	PVOL	
قيود التخزين	STOR	
القدرة على تحليل النظام	ACAP	خصائص شخصية
التواصل الشخصى	PCON	
خبرة المبرمجين فى مجال المشروع	PEXP	
قدرة المبرمجين	PCAP	
خبرة المحللين فى مجال المشروع	AEXP	
الخبرة فى استخدام أدوات التطوير	LTEX	
مدى استخدام أدوات البرمجيات	TOOL	خصائص المشروع
ضغط الجدولة	SCED	
إمكانية العمل على عدة مواقع على الويب	SITE	

شكل (٤-٨) الخواص المؤثرة فى تحديد قيمة المجهود

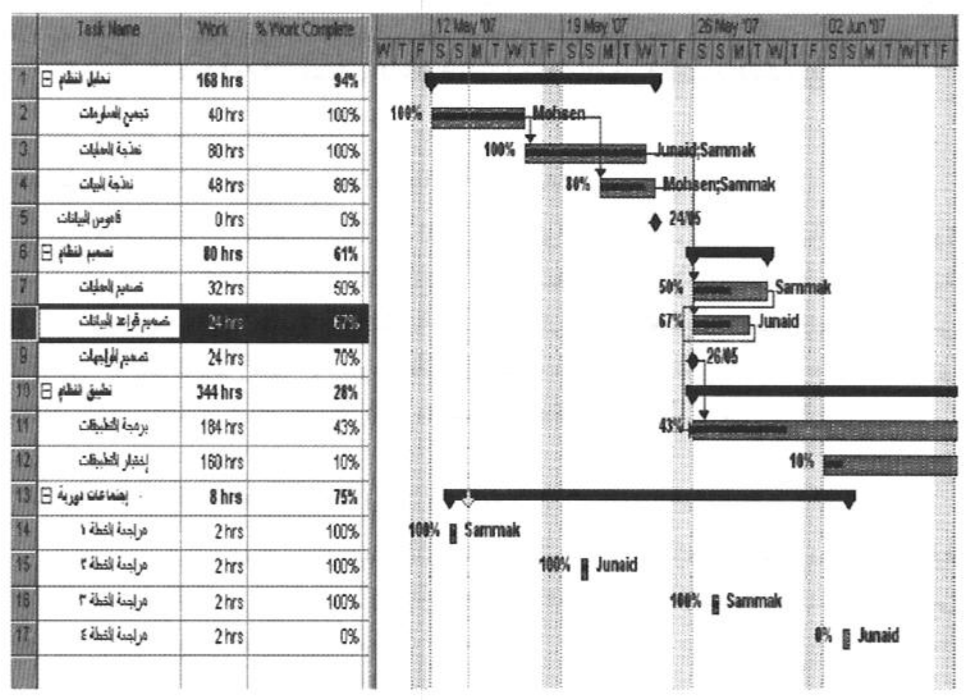
٤-٤ متابعة تقدم المشروع Tracking Project Progress:

يبدأ مدير المشروع فى متابعة تنفيذ مهام المشروع ومقارنة المنجزات مع المعلومات الموجودة بجدولة المشروع وكذلك مقارنة التكلفة الفعلية مع التكلفة المقدرة لكل مهمة. فمن الطبيعى أن تحدث عدة تغييرات فى كل من الجدولة وتكلفة الموارد المطلوبة وهذا يتطلب متابعة جيدة لإمكانية التحكم فى التغيير بصورة أفضل وبأقل تأثير على وقت تسليم المشروع وتكلفته المقدرة. وأحياناً تكون المعلومات المجمعة من متابعة الجدولة متعارضة مع المعلومات المجمعة من متابعة التكلفة،

حيث يكون إنجاز مهام المشروع متوافقاً مع الجدولة فى خطة المشروع ولكن مع تجاوز فى الميزانية المقدرة، وأحياناً يكون المشـــروع متوافقـــاً مع الميزانية المقدرة ويكون متأخراً عن جدولة المشروع. والآن سنعرض كلا المسارين بشىء من التفصيل.

٤-٤-١ متابعة جدول المشروع Tracking the Project Schedule:

يتم اســتخدام مخطط جانت GANTT chart لتسجيل معلومات الأجزاء المنفذة فى كل مهمة وكذلك تســجيل أوقات بداية ونهاية التنفيــذ الفعلى لكل مهمة والأفراد القائمين على إنجازها ونســبة مشــاركة كل فرد فى المهمة. تمثــل هذه المعلومات الجدولــة الفعليــة Actual Schedule للمشــروع مقابل الجدولة الأوليــة Baseline Schedule. يوضح شــكل (٤-٩) كيفية تسجيل معلومات المتابعة باستخدام مخطط جانــت، إذ يتم تظليل المهمة المنجزة بلون داكن حســب درجة اكتمال المهمة (تمت، لــم تبدأ بعد، تم إنجاز جــزء منها). ويتم أيضاً تســجيل معلومات عن حجم الإنجاز مقارنة بالموارد المستخدمة فى كل مهمة.



شكل (٤-٩) استخدام مخطط جانت لمتابعة جدولة المشروع

٤-٤-٢ متابعة الميزانية Tracking The Budget:

تعتبـــر عمليـــة متابعة الميزانية هـــى عملية مكملة لعملية متابعـــة الجدولة. فمن الممكن أن تتم متابعة التكلفة الفعلية للمهام المنجزة أو التى مازالت فى طور التنفيذ. بعـــد ذلك يتـــم تلخيص تلك المعلومـــات المجمعة وصياغتها فى شـــكل تقرير متابعة الميزانية ويتم تقديمه للإدارة العليا لاتخاذ القرار المناســـب. يوضح شـــكل (٤-١٠) كيفية حساب التكلفة المبدئية Baseline Cost، والتكلفة الفعلية Actual Cost، وكذلك التكلفة المتبقية Remaining Cost لكل مهمة.

	Task Name	Duration	Start	Finish	Predecessors	Resource Names	Fixed Cost	Baseline Cost	Actual Cost	Remaining Cost
1	⊞ تحليل النظام	10.5 days	Sat 12/05/07	Wed 23/05/07		Sammak;Mohsen	ر.س ٥٠٠،٠٠	ر.س ٢٧،٢٠٠،٠٠	ر.س ٤٦،٢٢٥،٠٠	ر.س ٢٣،٤٦٥،٠٠
5	قاموس البيانات	0 days	Thu 24/05/07	Thu 24/05/07		Junaid;Mohsen	ر.س ٥٠٠٠،٠٠	ر.س ٠،٠٠	ر.س ٠،٠٠	ر.س ٥٠٠٠،٠٠
6	⊞ تصميم النظام	4 days	Sat 26/05/07	Tue 29/05/07		Mohsen;Junaid;S	ر.س ٦٠٠،٠٠	ر.س ١٣،٦٠٠،٠٠	ر.س ١٦،٥٦٦،٠٠	ر.س ١٣،٩٣٤،٠٠
10	⊞ تطبيق النظام	15.5 days	Sat 26/05/07	Wed 13/06/07		Sammak;Mohsen	ر.س ١٠،٠٠٠،٠٠	ر.س ٥١،٦٠٠،٠٠	ر.س ٣٥،٤٤٤،٠٠	ر.س ٨٩،٣٩٩،٣٣
13	⊞ إجتماعات دورية	16.75 days	Sun 13/05/07	Sun 03/06/07		Sammak;Junaid	ر.س ١٠،٢٠٠،٠٠	ر.س ١٠،٦٠٠،٠٠	ر.س ٢،٩٠٠،٠٠	ر.س ٥٣،٥٠٠،٠٠

شكل (٤-١٠) حساب التكلفة المبدئية والفعلية لمهام المشروع

تحليل القيمة المكتسبة Earned Value Analysis:

يستخدم أسلوب القيمة المكتسبة لتحديد مدى حيد المشروع عن الميزانية المقدرة له فى خطة المشروع (Pressman, 2001). يستخدم هذا الأسلوب التقديرات الأولية فى الخطة المبدئية للمشروع والتكلفة الفعلية حتى الآن لتحديد الوضع المالى الحالى للمشروع. يتم تطبيق هذا الأسلوب على كل مهمة على حدة وعلى المشروع كله. يحتوى أسلوب تحليل القيمة المكتسبة على مجموعة المقاييس التالية (شكل (٤-١١)):

التكلفة المقدرة للأعمال المجدولة BCWS:

تمثل القيمة المكتســـبة للمهمة التى من المفترض أن تستخدم الموارد المخصصة لها فى جدول المشروع، وأن يتم إنجازها بالفعل قبل تاريخ التقييم، وتأخذ المهام التى لم تبدأ بعد القيمة المكتسبة صفراً.

تكلفة الأعمال المنجزة حسب الجدولة BCWP:

تمثل القيمة المكتسـبة للمهمة بناءً على نسبة الإنجاز المقدرة بالجدول التى تمت بالفعل قبل تاريخ التقييم.

التكلفة الفعلية للأعمال المنجزة ACWP:

تمثل مجمـوع التكلفة الفعلية للمهام المنجزة والتكاليف الثابتة وأى تكاليف أخرى يتم إدخالها يدوياً.

- التباين المجدول SV:

يمثـل الفرق بيـن تكلفة الأعمال المنجزة حسـب الجدولة والتكلفـة المقدرة فى الجدول (BCWP – BCWS)، وإذا كان قيمة SV تساوى الصفر، فهذا يعنى أن المهمة قـد تم إنجازها بالكامل أو أنها لم تبدأ بعد، وتأخذ قيمة سـالبة عندما تكون المهمة غير مكتملة.

- تباين التكلفة CV:

يمثل الفرق بين تكلفة الأعمال المنجزة حسـب الجدولة والتكلفة الفعلية للأعمال المنجزة (BCWP – ACWP). وهو مقياس يشـير إلى مدى زيادة أو نقصان التكلفة الفعلية للمهمة المنجزة عن التكلفة المقدرة بالخطة المبدئية للمشروع.

- التقدير عند الاكتمال EAC:

يمثـل التكلفـة المقدرة للمهمة عنـد اكتمالها بناءً على التقـدم الحالى فى إنجاز المهمة. وهو يساوى قيمة ACWP للمهام المكتملة بالفعل.

- الميزانية عند الاكتمال BAC:

يمثـل التكلفـة المجدولة للمهمة عند اكتمالها بناءً علـى التقدم الحالى فى إنجاز المهمة. وهو يساوى قيمة BCWP للمهام المكتملة بالفعل.

- التباين عند الاكتمال VAC:

يمثـل الفرق بين الميزانية عند الاكتمال والتقدير عند الاكتمال (BAC – EAC). وهو يســاوى قيمــة تباين التكلفة CV للمهام المكتملة، ويســاوى الصفر للمهام التى لم تبدأ بعد، وتكون قيمته ســالبة عند تجاوز التكلفــة الفعلية عن التكلفة المجدولة للمهمــة، وتكون قيمته موجبة عندما تكــون التكلفة الفعلية أقل من التكلفة المجدولة للمهمة.

	Task Name	.BCWS	BCWP	ACWP	SV	CV	EAC	BAC	VAC
1	⊞ تحليل النظام	ر.س. ٢،٤٠٠,٠٠	ر.س. ٢،٤٠٠,٠٠	ر.س. ٢،٦٨٥,٠٠	ر.س. ٠,٠٠	ر.س. -٢٨٥,٠٠	ر.س. ٣٠،٤٣٠,٠٠	ر.س. ٢٧،٢٠٠,٠٠	ر.س. -٣،٢٣٠,٠٠
5	⊞ قاموس البيانات	ر.س ٠٠٠٠	ر.س ٠٠٠٠	ر.س ٠٠٠٠	ر.س ٠٠٠٠	ر.س ٠٠٠٠	ر.س ٥٠٠٠٠٠	ر.س ٠٠٠٠	ر.س -٥٠٠٠٠٠
6	⊞ تصميم النظام	ر.س. ٠,٠٠	ر.س. ٠,٠٠	ر.س. ٠,٠٠	ر.س. ٠,٠٠	ر.س. ٠,٠٠	ر.س. ١٤،٨٠٠,٠٠	ر.س. ١٣،٦٠٠,٠٠	ر.س. -١،٢٠٠,٠٠
10	⊞ تطبيق النظام	ر.س. ٠,٠٠	ر.س. ٠,٠٠	ر.س. ٠,٠٠	ر.س. ٠,٠٠	ر.س. ٠,٠٠	ر.س. ٥٢،٦٠٠,٠٠	ر.س. ٥١،٦٠٠,٠٠	ر.س. -١،٠٠٠,٠٠
13	⊞ اجتماعات دورية	ر.س. ٤٠٠,٠٠	ر.س. ٤٠٠,٠٠	ر.س. ١،٣٠٠,٠٠	ر.س. ٠,٠٠	ر.س. -٩٠٠,٠٠	ر.س. ٥،٢٠٠,٠٠	ر.س. ١،٦٠٠,٠٠	ر.س. -٣،٦٠٠,٠٠

شكل (٤-١١) تحليل القيمة المكتسبة

٤-٥ إدارة الموارد البشرية Human Resources Management:

تركز عملية إدارة الموارد البشرية على أنشطة اكتساب الأفراد وتحفيزهم بالإضافة إلى عمليات تنظيم وإدارة الاتصال بينهم فى المشروع وكذلك تنمية قدراتهم وتدريبهم. تتســم عملية إدارة الموارد البشرية بالديناميكية نظراً للتغيير المستمر فى بيئة نظام الأعمال وهو تغيير يؤثر فى هيكل المؤسســة مما يتطلــب مهارات إبداعية من مدير المشــروع لإعادة هيكلة الموارد البشــرية. سوف نســتعرض الآن أهم أنشطة إدارة الموارد البشرية.

٤-٥-١ اكتساب وتحفيز الأفراد Acquiring and Motivating People:

يتم تشـكيل فريق العمل بالمشـروع من واقع خطة المشروع بمساعدة خطة إدارة الأفراد المصاحبة لها. وتعتبر عملية تحفيز واكتساب الأفراد هى جوهر عملية بناء أو تكوين فريق العمل. تتضمن عملية تكوين فريق العمل اختيار مدير الفريق وأعضاء الفريق حسب المهارات المطلوبة لإنجاز الأعمال بالمشـروع. وتعتبر عملية اختيار مدير المشروع من العوامل المؤثرة جداً فى نجاح المشروع. يجب أن تتوافر مجموعة من الخصائص المهمة فى شخصية مدير المشروع مثل:

- المقدرة على قيادة فريق العمل وتحقيق الأهداف المطلوبة.
- المقدرة على إقناع فريق العمل برؤيته.
- المهارات الأساسـية لأنشطة إدارة المشروع مثل التخطيط والجدولة والمتابعة وتقدير التكلفة وغيرها.
- مهارة اكتساب الأفراد وتحفيزهم.
- مهارة التنسيق والاتصال بين أفراد الفريق.
- المرونة فى التفكير ومهارة التفكير الاستدلالى.
- معرفة جيدة بنظم الأعمال.
- الثبات عند مواجهة الأزمات.

يقوم مدير المشـروع بتنفيذ مجموعة من المهام لتكوين فريق عمل فعال يناسـب مشروع الأعمال الحالى، وتتضمن المهام ما يلى:

- تشكيل فريق العمل.
- تحفيز أفراد الفريق.
- تنمية قدرات أفراد الفريق.
- تنسيق الاتصال بين أفراد الفريق وبين فرق العمل.

٤-٥-٢ تكوين فريق العمل Team Work Formation:

تحتاج مشـاريع البرمجيات الكبيـرة المعقدة إلى العديد من الخبرات فى مجالات مختلفة مثـل خبراء فى مجالات تقنية المعلومات مثـل التحليل والتصميم والبرمجة

والاختبار والتوثيق، بالإضافة إلى خبراء فى إدارة المشاريع، وخبراء فى عمليات دعم العميـل، وغيرها من الخبـرات. وبالطبع فإن مدير المشـروع يأمل أن يضم أفضل الخبـراء فـى كل مجال إلى فريق العمل الذى يحتاج إليه فى المشـروع، غير أن ذلك الأسـلوب لا يضمن بالقطع نجاح المشـروع إلا إذا كانوا يمتلكون مهارات شـخصية (بالإضافـة إلـى مهاراتهم الفنية العاليـة) تمكنهم من العمل معاً بـروح الفريق الذى يتطلب قدراً كبيراً من التعاون والمشـاركة والمرونة والتفاهم بينهم. ويعتبر ذلك من أهم أدوار مدير الفريق ذلك يشمل الواجبات التالية:

- تحديد المهام المطلوبة بدقة.
- تحديد الخبرات المطلوبة لإنجاز المهام.
- ترشيح عدد من الأفراد ذوى الخبرات المطلوبة.
- اختيار أفراد الفريق المتجانسين والأقدر على التفاهم والعمل معاً بتعاون تام.
- المتابعة والتوجيه المستمر لفريق العمل لضمان تحقيق الأهداف المطلوبة.
- التنسيق بين أفراد الفريق.
- حل التعارض الذى ينشأ بين أفراد الفريق فى أسرع وقت.

وفيما يلى عرض لبعض المهارات الفنية المطلوبة لمشاريع البرمجيات.

- تحليل وتصميم النظم.
- تصميم قواعد البيانات.
- برمجة التطبيقات.
- اختبار البرمجيات.
- توثيق التطبيقات.
- تصميم شبكات الحاسب.
- التصميم المعمارى للتطبيقات.
- تثبيت واستخدام أدوات البرمجيات.
- الدعم الفنى للعميل.

وذلك بالإضافة إلى مجموعة من المهارات الشخصية لأفراد الفريق مثل:

- التحفيز الذاتى والطموح الشخصى.
- التواصل الجيد مع الآخرين.
- القدرة على التركيز فى التفاصيل.
- المرونة والنضج فى التفكير.
- الإحساس بالمسؤولية عند الطوارئ.

٤-٥-٣ اتصالات المشروع Project Communications:

المقصود بالاتصالات هو تبادل المعلومات داخل المشروع، أى أنها تتضمن أنشطة إرسال واستقبال الرسائل بين المشاركين فى المشروع. تعتبر هذه العملية ذات أهمية كبيرة لنظم مشــاريع البرمجيات وتحتاج إلى إدارة جيدة لضمان الاســتفادة من تداول المعلومات داخل المشــروع. تركز عملية إدارة الاتصالات داخل المشــروع على عدة مفاهيم أو وجهات نظر مثل أشــكال الاتصالات وخطــوط الاتصال وعوامل الاتصال والاتصال لحل التعارضات، وسوف نستعرضها الآن بإيجاز.

أشكال الاتصالات Forms of Communications:

يتم تبادل الرسائل بعدة أشكال مختلفة مثل:

- شفهياً.
- كتابياً.
- باستخدام الصور.
- باستخدام الرموز.

يمكن أن تكون الرســالة ســرية أو عامة وتأخذ شــكلاً محدداً أو غير محددة الشكل، ويمكن أن تكون داخلية أو خارجية. تأخذ الاتصالات داخل المشروع عدة أشكال منها:

- اتصال عادى من شخص إلى شخص.
- اتصال بين الأفراد داخل غرفة الاجتماعات.

- محادثة هاتفية بين شخص وآخر.
- اتصال بين شخص وآخر عبر الإنترنت.
- اتصال عبر الفاكس.
- بريد عادى.
- اتصال عبر بريد إلكترونى.
- اتصال عبر صفحات الويب.

يتميز كل شكل من هذه الأشكال بمميزات خاصة تناسب ظروفاً معينة لإرسال واستقبال الرسائل. وفى الحقيقة يتم استخدام معظم هذه الأشكال فى أثناء إجراء الاتصالات داخل أى مشروع، مع الأخذ فى الاعتبار أن أكثر هذه الأشكال استخداماً هو أسلوب الرسائل الإلكترونية.

خطوط الاتصال Lines of communications:

تهتم عملية إدارة الاتصالات بالمشروع بتقليل خطوط الاتصال بين الأفراد وبين فريق العمل بقدر الإمكان. فمن الطبيعى أن تؤدى كثرة خطوط الاتصال إلى صعوبة فى عملية إدارة الاتصالات. تتطلب عملية إدارة الاتصالات تنظيم هيكل الاتصالات من خلال تنظيم الأفراد فى شكل فرق عمل وتحديد أسلوب الاتصال بينهم فى شكل هرمى. يتضمن الشكل الهرمى للاتصالات عدة خطوط اتصال فى الاتجاهين الرأسى والأفقى. ففى الاتجاه الأفقى توجد خطوط اتصال بين الأفراد، ويعتمد عدد الخطوط على عدد الأفراد فى هذا المستوى، إذ تزداد عدد خطوط الاتصال بشكل كبير جداً مع زيادة عدد الأفراد. ولتقليل عدد خطوط الاتصال يتم تجميع الأفراد فى شكل مجموعات عمل جزئية وتقتصر الاتصالات بينهم على الاتصال بين منسقى الفرق.

عوامل الاتصال Communication Factors:

من المعروف أن كثرة المعلومات عن اللازم أو قلتها عن الضرورى تؤدى إلى مشاكل كبيرة داخل المشروع. فيجب إمداد فرق العمل بالمعلومات المتعلقة بالمهام التى يقومون بإنجازها. ويجب إمداد باقى المشاركين فى المشروع بمعلومات عامة عن المنتجات والاتجاهات الخاصة بالمشروع حتى يكون لديهم فكرة واضحة عن

الوضع الحالى للمشـــروع. وهناك عامل آخر يخص الاتصـــال الأفقى بين الأفراد أو مجموعات العمل وهو التأكد من وصول المعلومات بشكل دقيق، ويتطلب ذلك الاهتمام بعملية اختيار الأفراد المسؤولين عن توصيل المعلومات. وينطبق المفهوم نفسه على الاتصـــالات التى تتم خلال اجتماعات العمل. فهنـــاك اجتماعات عمل دورية خاصة بفريق العمل للتشاور وتبادل الآراء والمعلومات. وهناك اجتماعات عامة تشمل جميع المشـــاركين فى المشـــروع وتتضمن معلومات عامة عن مشـــروع الأعمال والمنجزات الحالية.

الاتصال لحل التعارضات Communication for Conflict Resolution:

من الطبيعى أن يكون هناك كثير من التعارضات داخل المشـــروع يتم التغلب عليها غالباً من خلال الاتصال الجيد بين الأفراد أو فرق العمل. هناك عدة طرق تستخدم للتغلب على التعارضات الموجودة بالمشروع عن طريق الاتصال مثل:

الإجبار Forcing:

يتم إجبار أحد فرق العمل المتضاربة على تنفيذ أحد الحلول وذلك على مســـؤولية مديـــر المشـــروع؛ إذ إنه هو الـــذى اتخذ القرار باختيار هذا الحـــل. وهنا يجب على الفريق المتعارض أن يلتزم بتنفيذ ما تم إقراره من مدير المشروع.

التسوية Compromise:

هو أسلوب يتطلب الاتصال وجهاً لوجه بين المتعارضين ومحاولة الوصول إلى حل وسط يرضى الطرفين.

المواجهة Confrontation:

هى أكثر الطرق فعالية فى حل التعارضات، إذ يتم عرض الحقائق ومناقشة التبريرات الخاصة بكل مقترح، من ثم يتم غالباً الوصول إلى حل للمشكلة والاتفاق عليه.

التهدئة Smoothing:

هو أســـلوب مؤقت لحل المشـــكلة وذلك بتقليل أهمية الأجـــزاء التى تحتوى على تعارضات والتركيز على الأجزاء التى لا تحتوى على تعارضات. أحياناً تتلاشـــى هذه

التعارضات مع التقدم فى إنجاز المهام بالمشروع. وأحياناً أخرى تعود تلك التعارضات بشدة أكبر.

الانسحاب Withdrawal:

يتم فى هذا الأسلوب حل مشكلة التعارضات بانسحاب أحد الفرق المتعارضة من مناقشـــة موضوع التعارض مرة أخرى. ويعتبر هذا الأســلوب هو أسوأ الحلول لكونه يؤدى إلى تقليل خطوط الاتصال الضرورية بين فرق العمل بشـــكل خطير يؤثر ســلباً فى نجاح المشروع.

٤-٥-٤ تنمية فريق العمل Team Work Development:

لا ينبغى ترك الفريق، بعد تشـــكيله، منعزلاً فى أثناء العمل، ولكن يجب على مدير الفريق أن يقوم بمتابعة أفراد الفريق فى أثناء تأدية مهامه ويتدخل فى الوقت المناسب بإجراء تعديلات ضرورية على نشاطات المشروع لضمان نجاحه. نعرض الآن لبعض النشاطات التى يقوم بها مدير المشروع لمساعدة الفريق فى إنجاز أعمالهم بنجاح:

- التأكد من توافر القدر الكافى من الاتصال بين أفراد الفريق.
- التأكد من أن أفراد الفريق يتعاملون معاً باحترام.
- التأكد من أن كل فرد يتفهم دوره وواجباته بوضوح تام.
- التأكد من فهم كل أفراد الفريق لأهداف المشروع ودعمها.
- التأكد من توافر القدر الكافى من الاتفاق بين أفراد الفريق على العمليات المنجزة.
- التأكد من أن أفراد الفريق ليسوا من مدمنى التقاعس.

يتميــز فريق العمل المتجانس بالقدرة على تجاوز الخلافات بينهم بأنفســهم وحل التعــارض فى وجهات النظر أو الفروق فى المســتويات بينهــم، وبعد فترة يصبحون أكثر ترابطاً وتفاهماً مما يؤدى إلى زيادة ملحوظة فى كفاءتهم وإنتاجهم. ويجب على مدير الفريق أن يعطى أعضاء الفريق الفرصة والوقت لزيادة الارتباط والتفاهم بينهم ويتدخــل فقط فى المشــكلات الكبيرة أو التى تنتج من خــارج الفريق. ويمكن أيضاً لمديــر الفريق أن يقوم بعمل لقاءات عمل دورية لتبادل الأفكار ودراســة المشــكلات وتقريــب وجهات النظر للوصول إلى أفضل الحلــول. كل ذلك يعود بالنفع على أفراد الفريق لتنمية مهاراتهم وعلى المشروع فى النهاية.

٤-٦ إدارة المخاطر Risk Management:

مـــن النادر أن يخلو أى مشـــروع برمجيات من تعرضه لنـــوع من المخاطر. فهناك دائماً أسباب تؤدى إلى حدوث المخاطر مثل عدم الوصف الدقيق لأى جزء من أجزاء المشروع أو عدم الفهم الجيد له أو عدم دراسته جيداً أو عدم توثيقه جيداً أو ضعف الاتصال بين أفراد الفريق، وغيرها من الأســباب. فمن المعروف أن عملية تخطيط المشـــروع تتضمن العديد من الأنشـــطة بدءاً مـــن تحديد المتطلبات والمســـتلمات، مروراً بتحليل المهام وجدولتهـــا، وتحديد الموارد وتقديرها، وكذلك تحديد العمليات والمنهجيـــات والأدوات المطلوبة لإنجـــاز المهام. فمن الصعوبة أن يتم تنفيذ كل هذه الأنشطة بطريقة مثالية بدون أى مشكلات أو مخاطر. بالإضافة إلى أن هذه القائمة الطويلة من الأنشطة فى حد ذاتها تحمل مخاطر عديدة. وعلى هذا فمن الطبيعى أن يتم إجراء تعديلات على خطة المشروع وهو فى طور التنفيذ.

المخاطرة Risk هى مشـــكلة ناتجة من عدم التأكد من عملية أو أســـلوب أو موارد يتم استخدامها فى المشروع ولها تأثير سلبى فى المشروع، ويمكن التغلب عليها عند حدوثهـــا باعتماد خطـــة خاصة بالمخاطر المتوقعة. ومثال ذلـــك، عندما يقرر مدير المشـــروع استخدام تقنية جديدة فى المشروع تكون نتائجها غير مؤكدة وجديدة على فريق العمل، أو اســـتخدام موارد جديدة، أو انضمام عضو جديد لفريق العمل أو قلة الاتصال والتفاهم بين أعضاء الفريق، كل ذلك يؤدى إلى نوع من المخاطر من المحتمل أن تؤثر سلباً فى المشروع.

٤-٦-١ تحديد المخاطر Risk Identification:

كل مشـــروع يواجه مجموعـــة من المخاطر المحتملة حســـب طبيعته، وعلى مدير المشـــروع أن يكتشـــف أو يتوقع كل مواطن المخاطر الممكنة وأن يتعامل معها بجدية وصدق وعدم التهوين من شأنها. ويمكن لمدير المشروع أن يستعين بخبرات مديرين أكثر خبرةً منه فى نظم أعمال مشابهة. وبعد تحديد نوع الخطر، يجب وصفه بوضوح لفريـــق العمل ووصف تأثيره فى المشـــروع والأفعال المتطلبة للتغلـــب عليه. ويعتبر ضعف المفهوم فى أى مرحلة من مراحل إدارة وتنفيذ المشروع هو المصدر الرئيسى للمخاطر فى مشاريع البرمجيات. وفيما يلى عرض لأهم المصادر الرئيسية لحدوث المخاطر فى مشاريع البرمجيات:

- التفاؤل الزائد حول استخدام تقنيات جديدة.
- عدم فهم التأثيرات الحقيقية المؤثرة لبعض التقنيات.
- عدم الفهم التام لمتطلبات العميل.
- التغييرات المستمرة فى متطلبات العميل.
- عدم كفاية الموارد المطلوبة للمشروع.
- قلة خبرة أعضاء فريق العمل.
- التوقعات غير الحقيقية حول مهارة وإنتاجية بعض الموارد البشرية.
- تجاهل وجهات نظر وطلبات المستخدمين للمشروع.
- عدم وضوح ودقة المتطلبات الوظيفية لنظام الأعمال.
- التقييم غير الدقيق لموارد المشروع.
- إعداد خطة المشروع بطريقة متعجلة غير دقيقة.
- عدم إشراك باقى المساهمين فى المشروع فى أثناء إعداد خطة المشروع.

يجب على مدير المشـــروع أن يقوم بتحديــد قائمة بالمخاطر المتوقع حدوثها فى أثناء تنفيذ المشـــروع. تعتبر هذه القائمة غير نهائية ولكن يتم تحديثها باستمرار مع التقدم فى إنجاز مهام المشـــروع. ولتجهيز تلك القائمة يقوم مدير المشروع بالتركيز على الموضوعات الرئيسية التى تضمنها خطة المشروع وهى:

- مواصفات المنتجات والمستلمات للمشروع.
- قائمة المهام وجدولتها.
- الأهداف وطريقة قياسها.
- الموارد البشرية.
- العمليات والمنهجيات.
- الأدوات والمعدات.

فيقوم مدير المشـــروع بدراســـة كل بند من البنود الســـابقة بعنايـــة مع أكبر عدد ممكن من المشـــاركين فى المشروع وتبادل الآراء ووجهات النظر للوصول إلى القائمة المبدئية للمخاطر المتوقعة.

٤-٦-٢ تقييم المخاطر Risk Assessment:

يتم تقدير درجة المخاطر المتوقعة باستخدام طريقتين:

الأولى: احتمالية حدوث المخاطر Probability:

وهـــى تمثل مقياســـاً مدرجاً يبدأ مـــن (١٠٪) إلى (٩٠٪) وإذا وصلت النســـبة إلى (١٠٠٪) فإنها لا تعتبر مخاطرة إنما تصبح قيداً مؤكداً فى المشروع.

الثانية: التأثير السلبى فى المشروع Negative Impact:

وهو مقياس يتكون من خمس درجات هى:

- هائل/كارثى Catastrophic.
- حرج Critical.
- خطير Serious.
- محتمل Tolerable.
- هامشى Marginal.

يتم تمثيل قيمة التأثير الســـلبى فى المشـــروع من خلال تقدير قيمة تكلفة التغلب عليه، وبعد ذلك يتم حســـاب قيمة التعرض للمخاطرة لكل نوع من المخاطر المتوقعة كما يلى:

قيمة المخاطرة = احتمالية المخاطرة × قيمة التأثير السلبى

٤-٦-٣ معالجة المخاطر Risk Handling:

بعـــد عمليات تحديـــد المخاطر وتقييمهـــا، ننتقل الآن إلى نشـــاط مهم جداً وهو التخطيط لتخفيف آثار أو تجنب حدوث تلك المخاطر. ومثال ذلك أنه يحتمل حدوث مخاطر فى أثناء مهمة تكامل الأنظمة الفرعية للمشـــروع نتيجة وجود شـــخص واحد فقط عنده المقدرة على إنجاز تلك المهمة، وهنا يمكن لمدير المشـــروع اتخاذ أى من الأفعال التالية لتخفيف حدة أو تجنب هذه المخاطرة:

- توظيف شـخص إضافى يمتلك الخبرة المطلوبة واعتباره بديلاً ومسـاعداً للشخص الأساسى.

- عرض مزايا إضافية للشخص الوحيد المتوافر حالياً لضمان بقائه.

- استخدام طريقة أخرى لإنجاز المهمة بحيث لا تحتاج لتلك المهارة الخاصة.

ويعنـى هذا أنـه توجد غالباً عدة بدائـل لتخفيف المخاطر، ولكـن اتخاذ القرار باختيار أحدها يعتمد على عدة عوامل منها:

- سهولة أسلوب التخفيف.

- احتمالية نجاح الأسلوب.

- تكلفة الأسلوب.

فيمكن لمدير المشـروع بعد تحديد بدائل الحلول لتخفيف حدة المخاطر والتغلب عليهـا، أن يقـدر تكلفة كل بديل وبعد ذلك يتم اختيار البديـل الأقل تكلفة. ولكن مع الأخـذ فى الاعتبار احتمالية نجاح كل بديل، يمكن أن يؤدى ذلك إلى اختلاف القرار. فمـن الممكن أن يكون البديـل الأقل تكلفة هو أيضاً الأقـل احتمالية لنجاح تطبيقه، ويكـون بديل آخر أكثر تكلفة منه ولكن درجة احتمالية نجاحه أكبر، وهنا يكون القرار الأفضل هو اختيـار البديل الأكثر تكلفة والأكثر احتمالية للنجاح. ويمكن اسـتخدام الصيغة التالية التى تجمع بين العاملين لتحديد قيمة تكلفة تخفيف المخاطر:

تكلفة تخفيف المخاطرة = تكلفة الأسلوب * احتمالية فشل الأسلوب

خطة إزالة المخاطر Risk Removal Plan:

يقـوم مدير المشـروع بعد ذلك بإعداد خطة لإزالة المخاطـر بناء على المخاطر المحددة والمرتبة والأفعال والتدابير التى تم اختيارها لتجنب حدوثها. يوضح شـكل (٤-١٢) مثالاً لخطة إزالة المخاطر. تحتوى الخطة على قائمة المخاطر مرتبة حسب أولويـة حدوثها، وكذلك التاريخ المتوقع لإزالتها، مع ذكر الأفعال التى سـيتم اتخاذها لإزالة المخاطر.

الأفعال المطلوبة لإزالة المخاطر	التاريخ المتوقع لإزالة المخاطر	قائمة المخاطر (مرتبة)
إنجاز عمليات تخفيف المخاطرة	٢٠٠٦/٠٦/٣٠	مخاطرة ١
إنجاز المهمة رقم ٣	٢٠٠٦/٠٧/١٦	مخاطرة ٢
إنجاز المهمة رقم ٨، وإتمام عمليات تخفيف المخاطرة	٢٠٠٦/٠٩/٢١	مخاطرة ٣
.	.	.
.	.	.
.	.	.

شكل (٤-١٢) خطة إزالة المخاطر

يجب على مدير المشروع مراجعة خطة إزالة المخاطر باستمرار مع تقدم الإنجاز في المشروع، وذلك بحذف مخاطر من القائمة أو إضافة مخاطر أخرى طرأت حديثاً أو تغيير خصائص بعض المخاطر.

٤-٧ إدارة الجودة Quality Management:

يعتبر نشاط إدارة الجودة من الأنشطة التى تتضافر مع أنشطة إدارة المشروع وكذلك عمليات التطوير المختلفة للمشروع للتأكد من جودة البرمجيات المنتجة ومطابقتها لمواصفات الجودة المطلوبة. ولذلك فإن إدارة الجودة تكون غالباً مستقلة عن إدارة المشروع ولها فريق خاص غير فريق التطوير ولها ميزانية وخطة عمل مستقلة. هناك معايير قياسية دولية تستخدم لتقييم جودة البرمجيات للمؤسسات المنتجة لها وهى معروفة باسم "ISO 9000 Series".

تتضمن عملية إدارة الجودة الأنشطة التالية:

- إعداد خطة الجودة Quality Plan.
- مراقبة الجودة Quality Control.
- ضمان الجودة Quality Assurance.

٤-٧-١ جودة البرمجيات Software Quality:

تعتمد الجودة المطلوبة على طبيعة مشروع البرمجيات، التى تركز على نقاط جودة أكثر من غيرها. يتم تصنيف جودة البرمجيات إلى نوعين أساسيين هما:

- جودة المنتج Product Quality.
- جودة العملية Process Quality.

مـــع الأخذ فى الاعتبـــار أنه لا يمكن الحصول على منتـــج جيد بدون التأكد من جودة العمليات التى تمت لإنتاجه. ولهذا فإن إدارة الجودة تتطلب التأكد من جودة المنتج البرمجى خلال مراحل تطويره وليس بعد اســـتلامه فحســـب. تحتاج هذه العملية لاستخدام أداة إدارة التشكيل Configuration Management Tool لدعم عمليـــات تخزين ومعالجة بيانات النســـخ المطورة للمنتـــج البرمجى خلال مراحل تطويره. هناك العديد من الخصائص والمعايير التى تصف جودة المنتج البرمجى وهى (Ghezzi, 2003):

- الصحة Correctness.
- الثقة Reliability.
- المتانة Robustness.
- الأداء Performance.
- الاستخدام Usability.
- الفهم Understandability.
- قابلية الصيانة Maintainability.
- قابلية التطور Scalability.
- إعادة الاستخدام Reusability.
- قابلية النقل Portability.
- إمكانية التشغيل البينى Interoperability.
- الإنتاجية Productivity.
- الرؤية Visibility.

٤-٧-٢ مراقبة الجودة Quality Control:

تركــز عملية مراقبـــة الجودة على وجــود اختلافات بين مواصفـــات العمليات أو المنتجـــات البرمجية بالمقارنة بمواصفات المتطلبات. ولذلك فإن الهدف من عملية مراقبـــة الجودة هو إجراء العديد من الاختبارات علـــى العمليات التى تؤدى إلى إنتاج المنتجات البرمجية للحد من المشكلات التى تهدد جودة المنتج فى وقت مبكر. فقد أكدت دراســـة ســـابقة لشـــركة IBM أن تكلفة تصحيح الأخطاء الناجمة عن عمليات الاختبـــار ومراقبة الجودة تختلف بشـــدة حســـب المرحلة التى تم اكتشـــاف الأخطاء فيهـــا. فهى تكون قليلة عند اكتشـــافها مبكراً وتتراكم وتزداد بشـــكل غير خطى عند اكتشـــافها فـــى مراحل متأخرة. وعلـــى هذا فإن عملية الاختبـــار يجب أن تتم خلال مراحـــل التطوير المختلفة ويتم جدولتها بشـــكل منفصل ويقوم بها فريق مؤهل وتكون لها ميزانية منفصلة. تستخدم عملية مراقبة الجودة العديد من أساليب الاختبار التى تتضمن معظم المعايير التى تم سردها سابقاً لوصف جودة المنتج البرمجى. من أهم هذه الاختبارات ما يلى:

١- اختبار الصندوق الأسود Black Box Test:

نـــوع من الاختبارات يقوم على مبدأ عدم الاهتمام بمحتوى الوحدة البرمجية تحت الاختبار والتعامل معها باعتبارها صندوقاً أسود. ولذلك فإن هذا النوع من الاختبارات يناســـب مجموعة المستخدمين للنظام. تبدأ عملية الاختبار بتغذية الوحدة البرمجية ببيانات الاختبار الموجودة فى حالات الاختبار المعدة مســـبقاً، وبعد ذلك يتم مقارنة النتائـــج بما هو متوقع فى حالات الاختبار. يهتم هذا النوع من الاختبارات باعتبارات صحة وجودة المتطلبات الوظيفية للمنتج البرمجى وهو يعتبر مؤشـــراً لقبول أو رفض المنتج من ناحية مســـتخدمى النظام. يســـتخدم اختبار الصندوق الأســـود لاكتشاف الأخطاء التالية:

- وظائف غير صحيحة أو ناقصة.
- أخطاء فى واجهات استخدام النظام.
- أخطاء فى هياكل البيانات أو قواعد البيانات.
- أخطاء فى الأداء.

٢- اختبار الصندوق الأبيض White Box Test:

على النقيض من اختبار الصندوق الأسود، فإن اختبار الصندوق الأبيض يركز على اختبار محتوى المنتج البرمجى. ويقوم بتنفيذه مجموعة من المتخصصين فى اختبار البرمجيات. يبدأ المختبر عادة بمراجعة نماذج التصميم ومواصفات المتطلبات وبعد ذلك يبدأ بدراسة الأكواد وتتبع الأخطاء الموجودة وتصحيحها. ولذلك فإن طبيعة اختبار الصندوق الأبيض تعتمد على المراجعة والتدقيق والفحص المستمر لمراحل تطوير المنتج البرمجى بهدف اكتشاف الأخطاء وتصحيحها فى مرحلة مبكرة. جدير بالذكر أن هذا النوع من الاختبارات يعتمد بشكل أساسى على خبرة أفراد فريق الاختبار فى استخدام أساليب الاختبار بشكل جيد.

٣- الاختبار الآلى Automated Test:

نوع من الاختبارات يتم إجراؤه بشكل آلى باستخدام أدوات اختبار البرمجيات، وهو أسلوب يناسب التطبيقات الكبيرة الحجم التى يتم إعدادها بشكل قياسى باستخدام أدوات هندسة البرمجيات المدعمة بالحاسب CASE Tools.

خطة الاختبار Test Plan:

تعتبر خطة الاختبار جزءاً أساسياً من خطة إدارة الجودة وهى تحدد أنشطة جدولة عملية الاختبار وإعداد حالات الاختبار وتحديد الميزانية المطلوبة لتنفيذ الاختبار. تتضمن عملية التخطيط للاختبار الأنشطة التالية:

- تحديد المهام التى سيتم اختبارها.
- تحديد الأفراد الذين يقومون بتنفيذ الاختبار.
- توزيع مهام الاختبار على الأفراد.
- إعداد حالات الاختبار من واقع مواصفات متطلبات النظام.

تحتوى كل حالة اختبار على مواصفات الأجهزة والبرمجيات التى تناسب عملية الاختبار وكذلك المحاولات المقترحة لتنفيذ الاختبار. يتم تحقيق هذه المواصفات فى شكل سيناريو مكتوب يحتوى على جميع مواصفات حالات الاختبار وخطوات إجراء الاختبار وكذلك نقاط التحقق Verification Points.

٤-٧-٣ تأمين الجودة Quality Assurance:

تهدف عملية تأمين جودة البرمجيات إلى التحقق من مطابقة المنتجات البرمجية لمعايير الجودة التى تم إقرارها فى أهداف المشروع. يتم ذلك من خلال مراقبة خطة المشـــروع كاملةً مثل الجدولة والميزانية وتخصيص الموارد واســـتخدامها، بالإضافة إلى مراقبة إجراءات تطوير المنتج البرمجى. يتم اســـتخدام مجموعة من الأســـاليب التى تحقق تأمين جودة البرمجيات مثل:

١- قوائم التدقيق Checklists:

تحتوى قائمة التدقيق على مجموعة من الأســئلة والمواصفات التى تســـتخدم فى عمليات التحقق من جودة العملية أو المنتج البرمجى، وهى بذلك تمثل معايير الجودة المبدئيـــة للمنتج البرمجى. يمكن تجهيز قائمة التدقيق من خلال مجموعة من نقاط التدقيق المرتبة حســـب الأولوية وأهميتها للعمليـــة البرمجية. تخضع قائمة التدقيق للمعايير القياســـية الخاصة بهندســـة البرمجيات وهى تضمـــن مطابقتها لمواصفات الجودة العالمية ISO.

٢- مراجعات Reviews:

تتـــم مراجعة العمليـــات والمنتجات البرمجيـــة من خلال اجتماعـــات عمل تضم المطورين للعمليات التى يتم مراجعتها ومراجعين وقائد فريق المراجعة، وذلك للتأكد من جـــودة العمليات المنتجة. يتم تزويد أعضاء فريـــق المراجعة بالعمليات المنجزة لمراجعتهـــا وتدوين الملاحظـــات عليها قبل الاجتماع. يبدأ مطـــور العملية التى يتم مراجعتها بعرض ســـريع للإجراءات المنجزة مع الســـماح لأعضاء المراجعة بالتعليق عليها. يتم تســـجيل جميع الملاحظات فى وثيقة المراجعة التى يتم تســـليمها لمطور العمليـــة فى نهاية الاجتماع ليقوم بإجراء التصحيحـــات المطلوبة. بعد انتهاء عملية التصحيـــح يتم إبلاغ قائد فريق المراجعة بذلك ليقوم بمراجعتها واتخاذ القرار بقبول العملية أو الإعداد لاجتماع مراجعة آخر.

٣- تدقيقات Audits:

يعتبر أسلوب التدقيق أكثر شـــمولاً من أسلوب القوائم وأسلوب المراجعة، إذ يبدأ بعملية تجهيزية تقوم بدراســـة المنتج المطلوب تدقيقه، وبعد ذلك يتم إعداد مجموعة

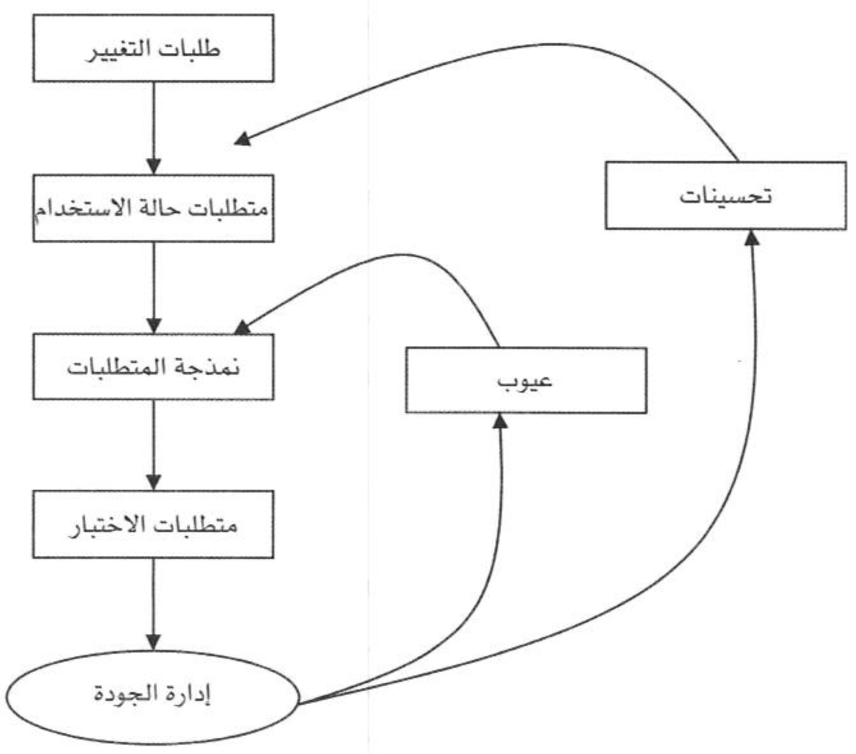
من الاستفســـارات فى شكل قائمة تدقيق Checklist يتم فحص العملية أو المنتج من خلالهـــا. يتبع ذلك إجراء مقابلات مع فريق التدقيق، إذ يتم تجميع معلومات التدقيق التـــى تم تدوينها فى أثنـــاء الفحص ومراجعتها، وبعد ذلك يتـــم توثيق خلاصة عملية التدقيق فى شكل تقرير فحص للمنتج البرمجى يحتوى على جميع الملاحظات الفنية على المنتج.

٤-٨ إدارة التشكيل والتغيير Change and Configuration Management:

تهتم عملية إدارة التشـــكيل والتغيير بتنظيم عمليـــة تطوير المنتج البرمجى وإدارة فريـــق العمل القائم على عملية التطوير. تشـــمل هذه العمليـــة مراحل تطوير المنتج البرمجـــى وما يتعرض له من تغييرات. يوضح الشـــكل (٤-١٣) المفهوم العام لعملية إدارة التشـــكيل والتغيير. يؤدى طلب التغيير غالباً إلـــى تعديل فى مواصفات حالات الاســـتخدام وهو يـــؤدى بدوره إلى تعديل نمـــاذج المتطلبات. وبعد ذلـــك يتم اختبار صلاحيـــة المتطلبات بعد إجـــراء التعديـــلات والتحقق من جودة نمـــاذج المتطلبات وتصحيحها أو إجراء تحسينات عليها وإعادة اختبارها وتقييمها.

٤-٨-١ التغيير فى المتطلبات Requirements Change:

تتســـم معظم مشـــاريع البرمجيات بإمكانية طلب المســـتخدم إجراء تغييرات فى متطلبات النظام فى أثناء عملية التطوير. أحياناً تكون بعض التغييرات جوهرية وذات تأثير مباشـــر فى خطة المشـــروع، وأحياناً تكون التغييرات يسيرة يمكن إدراجها فى متطلبات النظام من غير تأثير واضح فى خطة المشروع. ولهذا فإنه يجب التعامل مع طلبات التغيير بشكل رسمى من خلال عملية مخصصة لدراسة التغيير المطلوب ومدى تأثيره فى خطة المشروع الحالية ومن ثم قبوله أو رفضه. تبدأ هذه العملية بصياغة مواصفات التغيير المطلوب فى شـــكل طلب تغيير متطلبات. يحتوى هذا الطلب على بعض التوصيات حـــول التغيير المقترح بالإضافة إلى البيانـــات الخاصة بمواصفات التغييـــر وتأثيره فى خطة المشـــروع من حيث الوقت والميزانية. يتم دراســـة الطلب مـــن قبل مجموعة من الفنيين وتقييم التغيير المقترح من ناحية تأثيراته المتوقعة فى المشروع، ومن ثم تقرير مدى إمكانية تحقيق التغيير أو التوصية برفض طلب التغيير. يتم رفع هذا التقرير إلى مدير المشـــروع للموافقة على التوصيات أو إعادة الدراســـة مـــرة أخرى. وفى حالة الموافقة عليه يتم إصدار تقرير بمواصفات التغيير المطلوب يتضمن مجموعة القيود ومعايير الجودة التى تحكم عملية تطبيق التغيير.



شكل (٤-١٣) مفهوم عملية إدارة التشكيل والتغيير

٤-٨-٢ إصدارات المنتج Product Versions:

تؤدى عمليات التطوير خلال المراحل المختلفة إلى إنتاج مستلمات معينة سواء فى شكل نماذج أو برامج أو واجهات استخدام. ويؤدى أسلوب التطوير بالتكرار والزيادة لإنتاج أكثر من نسخة للمستلم أو المنتج. يتم تخزين جميع نسخ منتجات النظام فى مستودع رئيسى Repository ويتم التعامل معها من خلال إدارة نظام إدارة التشكيل Configuration Management system. يمكن التعديل على هذه النسخ فى أثناء تنفيذ طلبات التغيير فى المتطلبات ليتم الحصول على نسخ أخرى مطورة للمنتج، وهكذا حتى يتم الحصول على الإصدار الأخير للمنتج البرمجى. تكمن الاستفادة من المستودع فى إمكانية استخدام إحدى النسخ البديلة للمنتج فى مرحلة سابقة واستكمال عمليات التطوير عليها لإصدار نسخة مطورة للمنتج البرمجى. وبعد انتهاء عمليات الاختبار للنسخة المطورة للمنتج فإنه يتم الانتقال إلى مرحلة إصدار المنتج،

إذ يأخذ رقم إصدار ويتم تثبيته عند العميل لبدء التشــغيل الفعلى. تتضمن عمليات نشر وتثبيت الإصدار للمنتج البرمجى المستلمات التالية:

- ملفات تنفيذية Executable Files.
- برامج قواعد البيانات على الخادم Server Database Programs.
- ملفات قواعد البيانات Database Files.
- ملفات التشكيل Configuration Files.
- ملفات التوثيق System Documentations.
- ملفات المساعدة Help Files.
- تعليمات التثبيت Installation Instructions.

٤-٨-٣ العيوب والتحسينات Defects and Enhancements:

تظهر عيوب المنتجات البرمجية فى عدة مراحل فى أثناء عمليات المراجعة أو فى أثناء الاختبار أو فى أثناء التشغيل الفعلى عند العميل. وتكون التحسينات على المنتج فى شــكل توصيات مقترحة بإضافة ســمات جديدة أو إجــراء تعديلات فى الإصدار الحالى لإنتاج إصدار جديد محســن يفى بالمتطلبات الجديدة ولا يحتوى على العيوب الموجودة بالنســخة الحالية للمنتــج. تكون هذه التوصيــات المقترحة نتيجة لطلب مباشــر من أحد المشــاركين فى النظام بإجراء تعديلات أو إضافة متطلبات للنسخة الحالية وذلك بدون حدوث أى عيوب أو مشكلات فى الإصدار الحالى للمنتج. ويمكن أيضــاً أن تكون التوصيــات المقترحة نتيجة لحدوث عيوب ومشــكلات فى الإصدار الحالى ويجب علاجها وإصدار نســخة محسنة للإصدار الحالى. تأخذ العيوب التى يمكن ظهورها فى المنتجات البرمجية عدة أشكال أو حالات فى أثناء عملية معالجتها مثل:

- عيوب جديدة New.
- عيوب تم حلها Resolved.
- عيوب غير مؤكدة Unconfirmed.
- عيوب تم ظهورها مجدداً Reopened.

- عيوب تم تخصيصها Assigned.
- عيوب معلقة Suspended.
- عيوب فى حالة انتظار Waiting.
- عيوب تم إغلاقها Closed.

يقوم فريق المراجعة والاختبار بتحديد نوعية العيوب الموجودة وتقييم درجة خطورتها على المنتج ويمكن تقييم درجة خطورة العيوب المكتشفة فى المستويات التالية:

- بديهية Trivial.
- بسيطة Minor.
- عادية Normal.
- رئيسية Major.
- حرجة Critical.
- عائقة Blocker.

وبعد تحليل العيوب الناجمة، يتم اتخاذ الأفعال المناسبة للتغلب عليها فى شكل مجموعة من الإجراءات، ويأخذ تقرير الأفعال المطلوبة أحد الأشكال التالية:

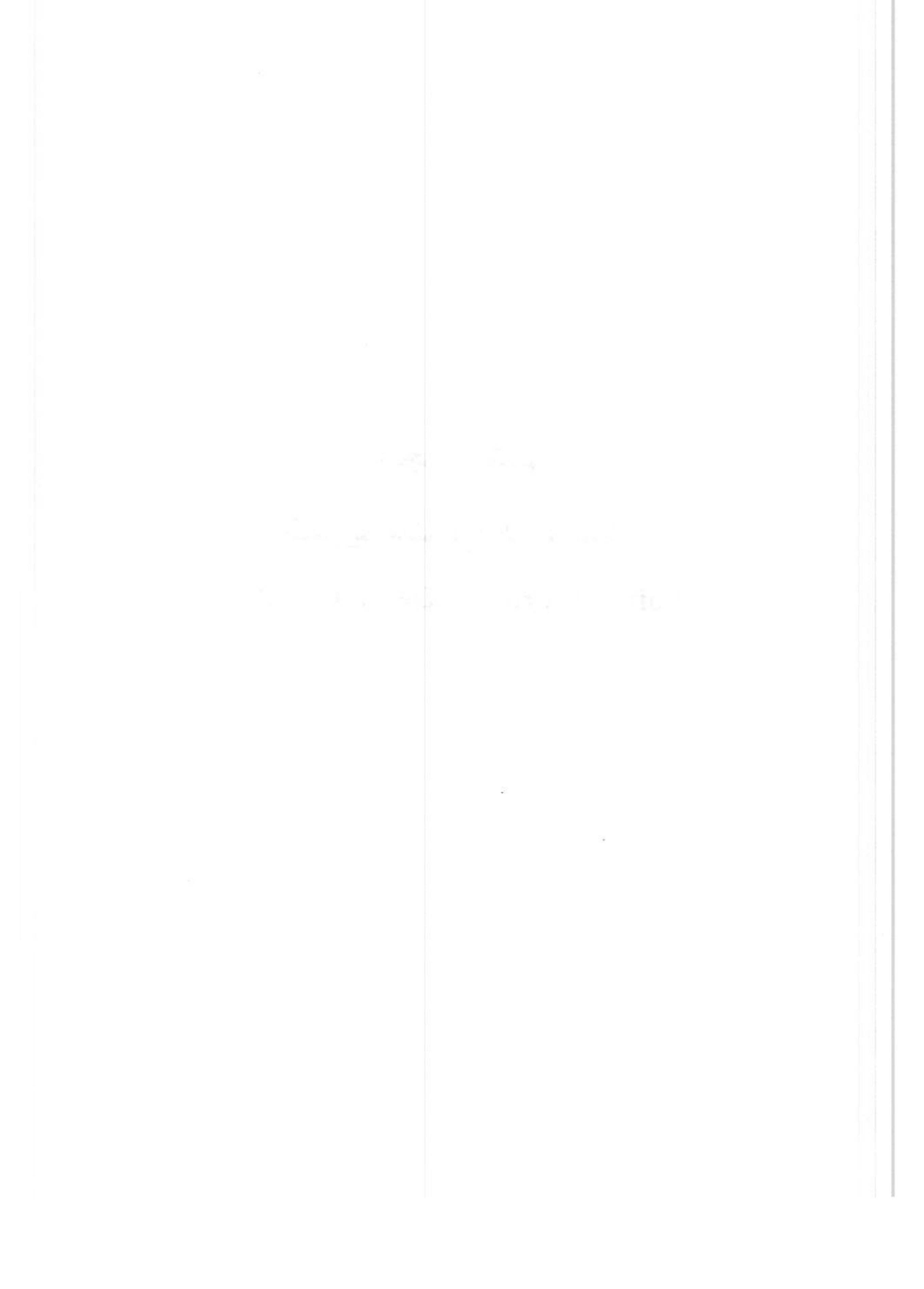
- غير صحيحة Invalid.
- مكررة Duplicate.
- جارى العمل فيها Workaround.
- تم حلها Fixed.
- لا يمكن حلها Wont fix.



الجزء الثانى

تطوير مشاريع البرمجيات

Software Projects Development



نقدم فـــى هذا الجزء عرضاً للعمليات التى يتم تنفيذها فى أثناء تطوير مشـــاريع البرمجيات. وســـوف نعتمد أحد الأســـاليب الحديثة لتطوير البرمجيات وهو أسلوب التكـــرار مع الزيـــادة Iterative with Increment الذى تم عرضه فى الجزء الأول من هذا الكتاب. يتميز هذا الأسلوب بمساعدة كل من فريق التطوير ومستخدمى النظام للمشـــاركة الفعالة خلال مراحل التطوير التى تضمن إنتاج برمجيات تحقق متطلبات المستخدمين. يتم ذلك من خلال إنتاج البرمجيات المطلوبة بالتدريج وذلك بعد تحديد أولويات متطلبات النظام من قبل المســـتخدمين. فتبدأ عمليات التطوير دائماً بتحليل المتطلبـــات الضرورية للنظام، ثم تصميـــم الحلول لها، ثم تطبيقهـــا واختبارها، وبعد ذلك يتم تســـليمها للمستخدم فى شكل نســـخة أولية للنظام. يقوم فريق المستخدمين بتجربة النســـخة الأولية للنظام وإبداء الملاحظات ومناقشتها مع فريق التطوير لإجراء التعديـــلات المطلوبة مبكـــراً. وبعد ذلك يتم إضافة متطلبات أخرى للنســـخة المطورة الحالية وإعادة عمليات التطوير حتى يتم الحصول على نســـخة أخرى موسعة تحتوى على وظائف إضافية للنسخة الحالية. يتم تكرار هذا الأسلوب حتى يتم إنتاج النسخة النهائية للنظام التى تتضمن جميع متطلبات النظام وذلك بمشـــاركة معظم المساهمين فى مشروع البرمجيات. يؤدى هذا الأسلوب إلى إنتاج برمجيات ذات جودة عالية تحوز ثقة العميل والمستخدم.

يشمل هذا الجزء عرضاً وافياً لعمليات تطوير النظم مثل:

- تحليل متطلبات النظام System Requirements Analysis.
- التصميم المعمارى للنظام Architectural System Design.
- تصميم قواعد البيانات Database Design.
- برمجة واختبار التطبيق Programming and Testing.
- تصميم واجهات المستخدم User Interface Design.

وســـوف نستعين ببعض التطبيقات والحالات الدراسية لدعم عملية عرض أنشطة تطوير مشاريع البرمجيات.

الفصل الخامس
تحديد المتطلبات
Requirements Determination

يتضمن أى مشـــروع برمجيات مجموعة من المتطلبات والاحتياجات التى تصف مـــا يجب أن يقوم به النظام وظروف التشـــغيل. يتم تصنيف المتطلبات إلى نوعين هما:

١- متطلبات وظيفية:

تصف مجموعة الخدمات والوظائف التى يجب أن يوفرها النظام.

٢- متطلبات غير وظيفية:

تصف مجموعة القيود التى تحكم تشغيل النظام.

تحتوى عملية تحديد متطلبات مشروع البرمجيات على العديد من الأنشطة مثل:

- تجميع المعلومات.
- تعريف المتطلبات الوظيفية وغير الوظيفية.
- تحديد أولويات المتطلبات.
- تحليل المتطلبات.
- توثيق المتطلبات.

وسوف نركز فى هذا الفصل على النشاط الخاص بعمليات تحليل المتطلبات.

٥-١ نموذج كائنات نظام الأعمال Business Object Model:

يعتبر نموذج كائنات نظام الأعمال BOM بمنزلة إطار العمل Framework الخاص بعمليات تطوير نظام الأعمال بالمؤسســـة. وهو يســـتخدم أيضاً وســـيلة اتصال فعالة بين فريق التطوير والمســـتخدمين فى أثناء عمليات التطوير. وكما ذكرنا ســـابقاً فإن اســـتخدام أســـلوب النمذجة المرئية يســـاعد كثيراً فى توصيل المفهوم فى أثناء عملية

تحليــل المتطلبات. وهو أســلوب يمكننا من وصف النظام من خــلال عدة منظورات ويصف جميع العلاقات والتفاعلات الموجودة داخل نظام الأعمال. يصف هذا النموذج مجموعة الكائنات الأساســية لنظام الأعمال والعلاقات بينها فى وضع السكون أو ما يسمى بالمنظور التركيبى Structural View لنظام الأعمال. ويصف كذلك تفاعل نظام الأعمال مع البيئة المحيطة به من خلال وصف مجموعة الفاعلين Actors وتفاعلهم مع نظام الأعمال من خلال مجموعة من حالات الاســتخدام Use Cases . ويمكننا أيضاً اســتخدام أســلوب التحليل الهيكلى للمتطلبات عن طريق اســتخدام مخطط السياق Context Diagram ومخططات تدفق البيانات (Data Flow Diagrams (DFDs.

٥-١-١ نظام الأعمال The Business:

تعتمــد صحة وجودة عملية تحديد وتحليل المتطلبات على مدى فهم فريق التطوير لنظام الأعمال. تشمل عملية الفهم عدة جوانب مهمة مثل:

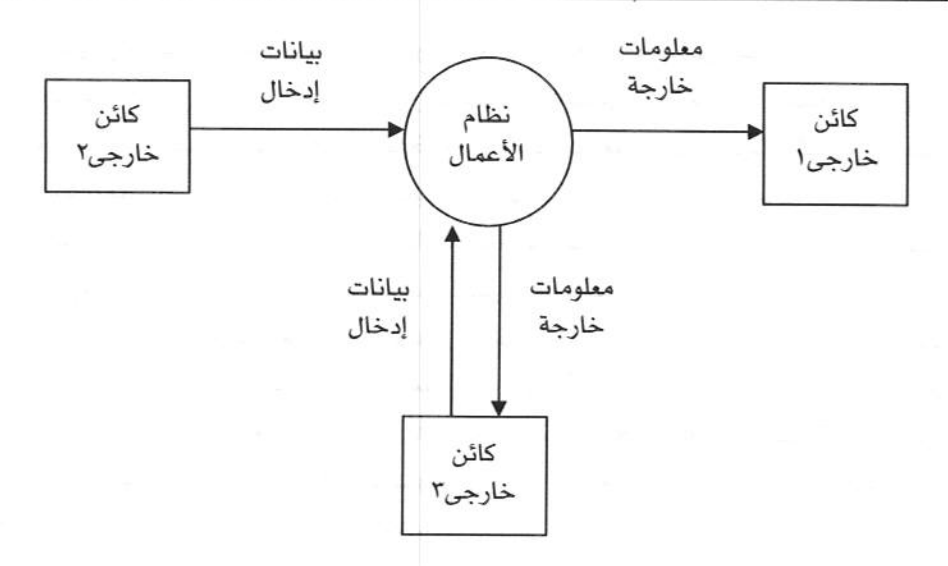
- أهداف نظام الأعمال Business Objectives.
- مجالات نظام الأعمال Business Domains.
- الهيكل التنظيمى لنظام الأعمال Organizational Structure.
- الأنظمة الفرعية داخل نظام الأعمال والتفاعلات بينها Interrelated Subsystems.
- مجموعة الفاعلين لنظام الأعمال Actors.
- المتطلبات الوظيفية لنظام الأعمال Functional Requirements.
- المتطلبات غير الوظيفية لنظام الأعمال Nonfunctional Requirements.

تتم عمليــة فهم نظام الأعمال وتحديد متطلباته من خــلال عقد لقاءات عمل مع كل من العميل ومستخدمى النظام. يقوم العميل أو صاحب نظام الأعمال أو المسؤول وصاحــب القرار بتحديد الأهداف العامــة لنظام الأعمال وتحديد مجموعة متطلبات النظام غير الوظيفية مثل: الجودة والكفاءة المطلوبة وزمن التنفيذ والميزانية المخصصة وطريقة التسليم ومعايير القبول ومتطلبات السلامة ومتطلبات أخرى قانونية وغيرها مــن المتطلبات غير الوظيفية. أما مواصفــات متطلبات نظام الأعمال الوظيفية فيتم

معرفتها وفهمها من خلال لقاءات دورية مجدولة مع مستخدمى النظام فى المستويات المختلفة بدءاً من مديرى العموم مروراً برؤساء الأقسام ووصولاً إلى المستخدم النهائى لمهام نظام الأعمال. يقوم كل مستخدم حسب موقعه ومسؤوليته بشرح ووصف طبيعة المهام التى يقوم بها ووصف للبيانات التى يستقبلها ومصدرها والمعلومات التى ينتجها والجهات التى تصدر إليها. بالإضافة إلى عرض للمشكلات الحالية بالنظام والمقترحات لحلها إذا أمكن وكذلك التحسينات المطلوبة من المستخدم على إجراءات تنفيذ المهام. يتم الاستعانة بمجموعة الوثائق والمستندات واللوائح والتقارير التى يستخدمها ويتعامل معها مستخدمو النظام فى أثناء إنجاز أعمالهم. ومن الممكن الاستعانة بأفراد أو جهات خارجية ذات خبرة فى مجال نظام الأعمال للتأكد من صحة ودقة ومصداقية المعلومات والمتطلبات المجمعة ومدى إمكانية تحقيقها فى الوقت المطلوب وبالميزانية المخصصة.

٥-١-٢ مخطط السياق لنظام الأعمال Business Context Diagram:

يصف هذا المخطط حدود نظام الأعمال العام داخل البيئة المحيطة به. يتم تمثيل ذلك من خلال نشاط عام يمثل نظام الأعمال العام وتفاعلاته المختلفة مع الكائنات الخارجية التى تتعامل مع نظام الأعمال. تمثل هذه الكائنات الخارجية مجموعة الأفراد أو المؤسسات أو الأنظمة الأخرى التى تتعامل مع نظام الأعمال. يتم تمثيل التفاعلات بين الكائنات الخارجية ونظام الأعمال من خلال مجموعة من تدفقات البيانات Data Flows . تمثل هذه التدفقات البيانات التى يستقبلها نظام الأعمال من الكائنات الخارجية وكذلك المعلومات التى يرسلها نظام الأعمال إلى الكائنات الخارجية. وعلى هذا الأساس يتم تمثيل نظام الأعمال فى مخطط السياق على شكل عملية أو نشاط عام يمثل مجموعة العمليات والإجراءات التى تتم على البيانات المدخلة للنظام لإنتاج المعلومات التى سيتم إمدادها إلى الكائنات المحيطة بالنظام للاستفادة منها. يوضح شكل (٥-١) الشكل العام لمخطط السياق لنظام الأعمال. نلاحظ أن بعض الكائنات الخارجية تكون مرسلة للبيانات فقط أو مستقبلة لها والبعض الآخر يكون مرسلاً ومستقبلاً للبيانات.



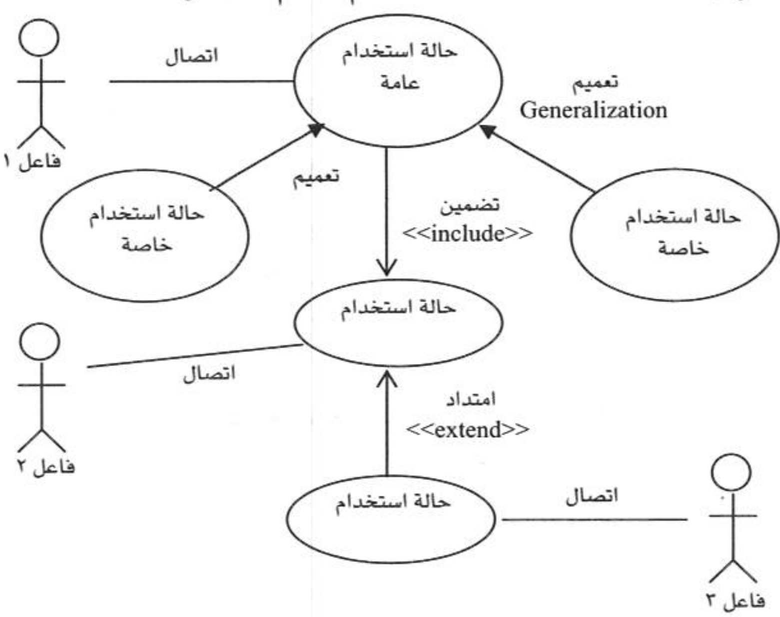
شكل (٥-١) مخطط السياق لنظام الأعمال

ومع أن مخطط السياق ليس من المخططات المعتمدة فى لغة UML، ولكن من الأفضل أن يتم البدء به فى عملية نمذجة المتطلبات، إذ إنه يعطى صورة واضحة ومفيدة لكيفية تعامل النظام مع البيئة المحيطة به من خلال مجموعة التفاعلات مع مستخدمى النظام.

٥-١-٣ مخطط حالات الاستخدام لنظام الأعمال Business Use Case Diagram:

يمثل مخطط حالات الاستخدام لنظام الأعمال العام المستوى الأول لتجزئة مخطط السياق لنظام الأعمال. يمكننا بعد ذلك تجزئة هذا المخطط العام إلى مجموعة من مخططات حالات الاستخدام الفرعية للنظام. وعلى هذا فهو يمثل حالات استخدام نظام الأعمال فى المستوى الأعلى فى شكل وحدات وظيفية رئيسية ذات حدود واضحة تمثل النشاطات الأساسية لنظام الأعمال. يحتوى نموذج حالات الاستخدام للنظام على وصف لمجموعة الفاعلين Actors الذين يتفاعلون مع حالات الاستخدام. يمثل الفاعل مجموعة الأشخاص أو الأنظمة التى تتعامل مع الخدمات التى تمثلها حالات الاستخدام. يمكن أن يمثل الفاعل الكائنات الخارجية لنظام الأعمال أو يمثل دوراً داخلياً مباشراً داخل النظام مثل مستخدمى النظام. ويحتوى نموذج حالات الاستخدام للنظام أيضاً على مجموعة من حالات استخدام النظام التى يمكن أن ترتبط معاً

بعلاقـات ارتبـاط Association أو علاقات تعميم Generalization. يوضح شـكل (٥-٢) نموذجاً عاماً لمخطط حالات استخدام لنظام الأعمال.



شكل (٥-٢) نموذج لمخطط حالات الاستخدام لنظام الأعمال

٥-١-٤ مخطط الأصناف لنظام الأعمال Business Class Diagram:

يوضح هذا النموذج مجموعة الكائنات الأساسـية لنظام الأعمال والعلاقات بينها. يسـهم كل كائن أساسى فى تقديم خدمة أو أكثر من الخدمات الأساسية التى يقدمها نظـام الأعمال. يتم تمثيل أصناف نظام الأعمال بشـكل مختلف عن أصناف المجال، فيتـم تمثيلها من خلال دائرة تحتهـا خط أفقى وبداخلها خط مائل. ومع أن الصنف يحتوى على مجموعة من الخصائص والعمليات ولكن لا يتم تمثيلها فى نموذج أصناف نظام الأعمال؛ لأنه يمثل الكائنات الأساسية فى المستويات العليا لنظام الأعمال وبشكل مجرد. هناك ثلاثة أنواع من العلاقات يمكن تمثيلها بين الأصناف العليا هى:

- الارتباط Association.
- التعميم Generalization.
- التجميع Aggregation.

يمكننـــا أيضاً توضيح درجة التعدديـــة Multiplicity لكل طــرف فى العلاقة بين الأصناف مثل علاقات:

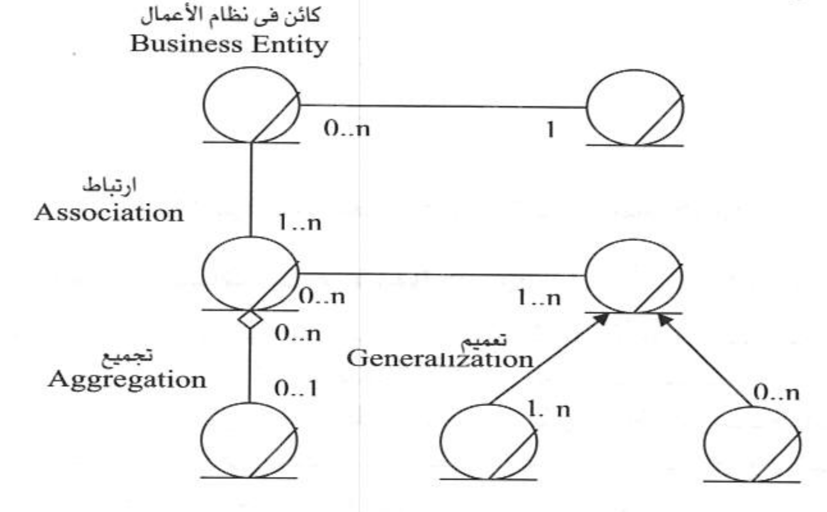
واحد- إلى - واحد One-to-One.

واحد- إلى - متعدد One-to-Many.

متعدد- إلى- متعدد Many-to-Many.

ويمكــن أن تكــون العلاقة اختيارية Optional عندما يكون من الممكن أن يشـــارك الكائــن فى العلاقة بإحدى حالاته أو لا يشـــارك بأى حالـــة. ويمكن أن تكون العلاقة إجبارية Mandatory بأن يشـــارك الكائن فى العلاقة بأكثر من حالة أو يشـــارك على الأقل بحالة واحدة.

يوضح شكل (٥-٣) نموذجاً لمخطط الأصناف لنظام الأعمال العام.



شكل (٥-٣) نموذج لمخطط الأصناف لنظام الأعمال العام

٥-٢ نموذج كائنات المجال Domain Object Model:

يعتبر نموذج كائنـــات المجال DOM هو أحد النماذج الفرعية لنموذج كائنات نظام الأعمـــال BOM. أى أنـــه يوجد دائماً نموذج واحد يصف كائنات النظام BOM وعدة نمـــاذج تصف كائنات المجـــال DOMs. ونجد أيضاً أن كل حالة اســـتخدام عامة فى

نمـــوذج كائنـــات النظام يتم تمثيلها من خـــلال نموذج كائنات لأحـــد مجالات النظام. والمجال هو أحد التطبيقات أو النظم الفرعية للنظام العام لمشـــروع الأعمال. والمجال له حدود واضحة متمثلة فى شـــكل مجموعة محددة من حالات الاستخدام والفاعلين المشـــاركين فيها. وعلى هذا فإن عملية النمذجة ســـيتم تطبيقها مباشرة على مستوى كائنات المجال.

٥-٢-١ المجال Domain:

كما ذكرنا سابقاً فإن أى نظام أعمال يحتوى على مجموعة من الوحدات الوظيفية الفرعية التى تتضمن مجموعة الخدمات والوظائف التى يقدمها نظام الأعمال. ومن الطبيعـــى أن تتبع كل وحدة فرعية أحد المجالات التـــى يتضمنها نظام الأعمال العام. فمثلاً يعتبر نظام الإجازات أحد المجالات الفرعية للنظام العام لشـــؤون الموظفين بأى مؤسســـة. وهو يحتوى على مجموعة الوظائف والعمليـــات وقواعد العمل التى تحكم عمليـــة منح الإجازات للموظفين وكذلك تفاعلها مع مجالات أخرى داخل النظام العام مثل تأثير الإجازات فى الرواتب والترقيات والابتعاث وغيرها من النظم الفرعية داخل نظام شؤون الموظفين.

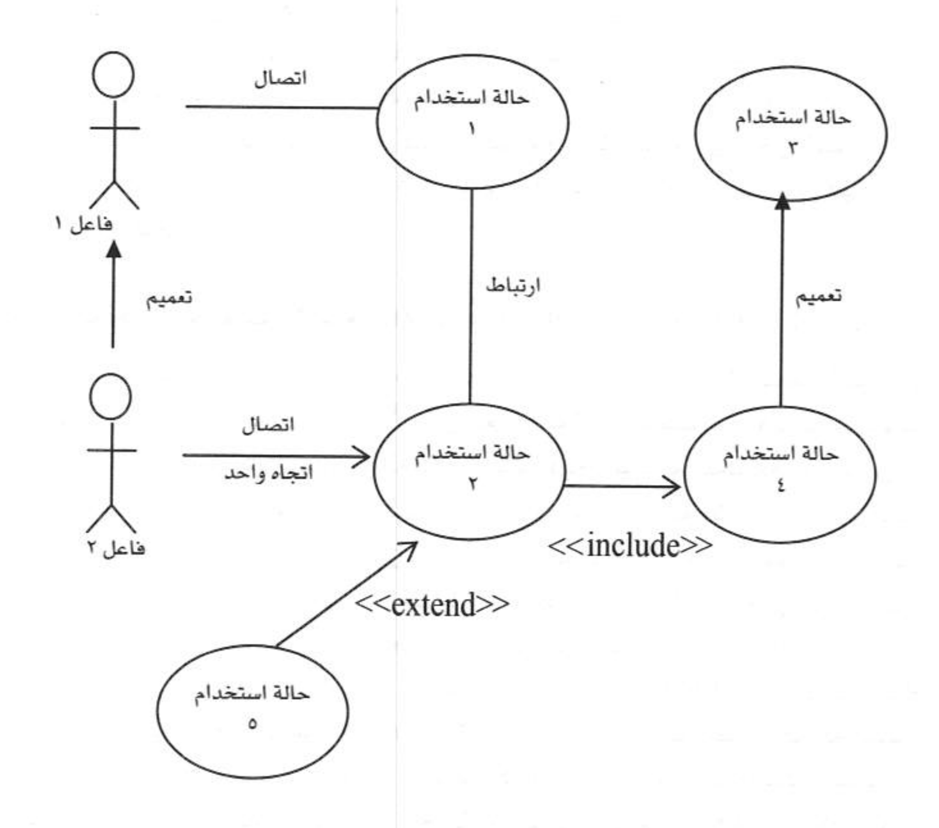
٥-٢-٢ مخطط حالات الاستخدام للمجال Domain Use Case Diagram:

تعتبـــر مخططات حالات الاســـتخدام لمجالات النظام بمنزلـــة الوصف التفصيلى لمخطط حالات الاســـتخدام لنظام الأعمال العام. يمثل كل مخطط تفصيلى مجموعة الوظائـــف لأحد مجالات النظـــام والعلاقات بينها وبين البيئـــة المحيطة به. تمثل كل حالة اســـتخدام Use Case وظيفة رئيســـية من وظائف أحد مجالات النظام وتتفاعل مـــع الفاعلـــين Actors الذين يصفون بيئة المجال. يمكـــن أن يحتوى المجال على عدة مخططات لحالات الاســـتخدام، وأحياناً يتم وصـــف جميع حالات المجال فى مخطط واحد. ترتبط حالات الاســـتخدام معاً فى شـــكل علاقات ارتبـــاط Association أو علاقـــات تعميم Generalization أو علاقات تضمـــين Inclusion أو علاقات امتداد Extension. ويرتبـــط الفاعل Actor بحالات الاســـتخدام مـــن خلال علاقة اتصال Communication. جدير بالذكر أنه لا يمكن الاعتماد على مخطط حالات الاستخدام لوصف وظائف المجال فحســـب، إنما يتم الاستعانة بملفات نصية Text Documents ترفق مع المخطط لتوثيق مواصفات حالات الاســـتخدام التى يتم تخزينها مع مخطط

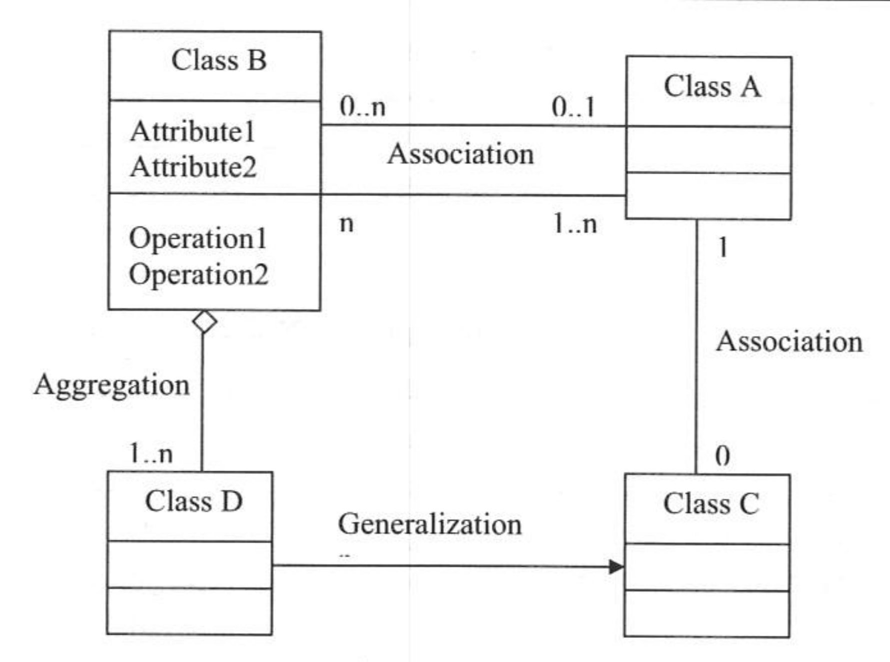
حالات الاســتخدام للمجال فى مســتودع النظام Repository. يوضح شــكل (٥-٤) نموذجاً لمخطط حالات الاستخدام لأحد مجالات نظام الأعمال العام.

٥-٢-٣ مخطط الأصناف للمجال Domain Class Diagram:

يمثل مخطط الأصناف للمجال المنظور التركيبى لأحد مجالات نظام الأعمال من خلال تمثيل أصناف المجال والعلاقات بينها فى شكل نموذج مرئى. يتضمن المخطط وصفاً لبعض الخصائص المهمة للأصناف خاصة التى تســهم فى عمليات الربط بين الأصنــاف وكذلك العمليات الرئيســية التى يقوم بها الصنف. ويوضح شــكل (٥-٥) نموذجاً لمخطط الأصناف لأحد مجالات نظام الأعمال العام.



شكل (٥-٤) نموذج لمخطط حالات استخدام المجال



شكل (٥-٥) نموذج مخطط الأصناف للمجال

تدعم أدوات النمذجة باستخدام لغة UML إمكانية تسجيل الخصائص والعمليات التى لم تظهر فى مخطط الأصناف بالإضافة إلى معلومات أخرى إضافية يتم تخزينها فى مستودع نظام الأعمال.

يوضح الشكل إمكانية أن تكون علاقة الارتباط Association أحادية الاتجاه Unidirectional أى أنه لا توجد أى حالة من حالات الصنف A يمكن أن ترتبط بحالة من حالات الصنف C ولكن الارتباط يكون مسموحاً فقط من الصنف C إلى الصنف A.

٥-٣ حالة دراسية Case Study:

سنحاول فى هذا الجزء تطبيق أسلوب التكرار مع الزيادة لتطوير التطبيق الخاص بإدارة البريد الإلكترونى Email Management (EM) وهو يمكن أن يكون جزءاً مهماً فى العديد من أنظمة المعلومات. يمكننا أيضاً استخدام هذا التطبيق على أنه نظام تسويق عبر الإنترنت من خلال البريد الإلكترونى Email Marketing الذى يستهدف عدداً كبيراً من العملاء حول العالم ويمدهم بالمعلومات التسويقية عن منتجات معينة

أو خدمات تقدم من خلال الإنترنت، وذلك باســتخدام أســلوب الرســائل الإلكترونية عبــر البريد الإلكترونى. يقوم النظام بتســجيل بيانات العملاء وعناوينهم الإلكترونية فى قاعدة البيانات، ثم يقوم بتجهيز الرســائل الترويجية للمنتجات والخدمات، ويقوم بإرســالها لجميع العملاء. يقوم النظام أيضاً بعمليات تحديث دورية للبيانات المسجلة فى قاعدة البيانات.

٥-٣-١ نموذج حالات الاستخدام Use Case Model:

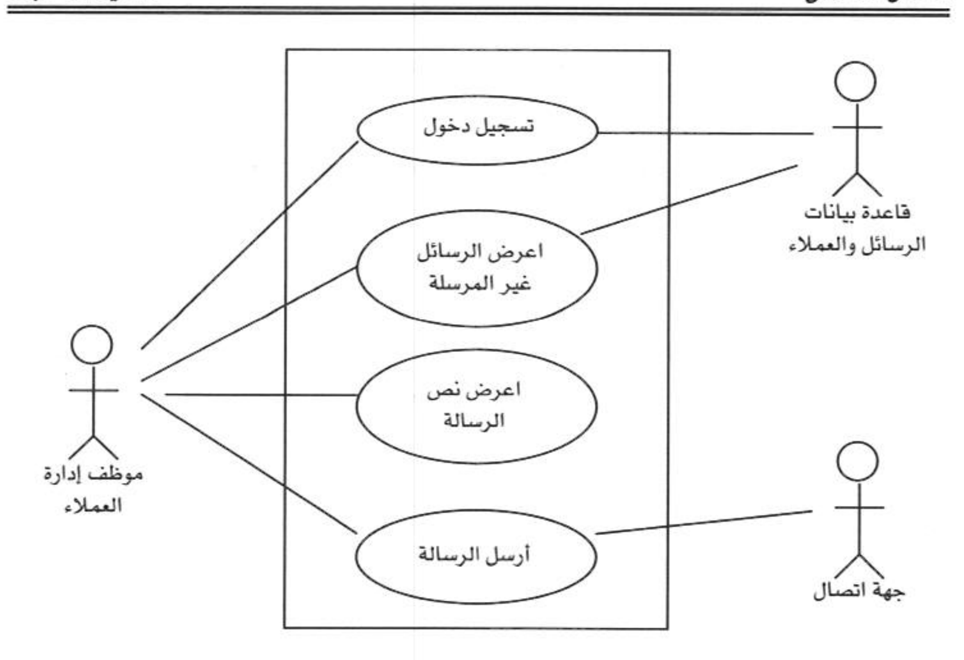
ســوف نفترض فى هذه الحالة الدراسية أن الرســائل الترويجية قد تم تجهيزها وتســجيلها من قبل فى قاعدة البيانات للنظام. وعلى هذا ســيكون الهدف من الحالة الدراســية هو إرسال الرســائل المخزنة فى قاعدة البيانات والاستعلام عنها. يمكننا توصيف متطلبات الحالة الدراسية من خلال مجموعة من حالات الاستخدام والفاعلين الذين يتعاملون معها. يوضح شكل (٥-٦) نموذج حالات الاستخدام لنظام إدارة البريد الإلكترونى EM.

يحتوى النموذج على أربع حالات استخدام هى:

- تسجيل دخول Login.
- أرسل الرسالة Send Message.
- اعرض نص الرسالة Display Message.
- اعرض الرسائل غير المرسلة View Unsent Messages.

بالإضافة إلى مجموعة الفاعلين:

- موظف إدارة العملاء Customer Department Employee.
- جهة اتصال Contact.
- قاعدة بيانات الرسائل والعملاء Production Database.



شكل (٥-٦) نموذج حالات الاستخدام لنظام إدارة البريد الإلكترونى

٥-٣-٢ وثيقة حالة الاستخدام Use Case Document:

على الرغم من أهمية استخدام النماذج المرئية لوصف متطلبات النظام إلا أنها لا تكفى لمعرفة المتطلبات بشكل مفصل. يتم توصيف محتويات حالة الاستخدام بشكل نصى من خلال وثيقة تسمى «وثيقة حالة الاستخدام» وتكون مرفقة بنموذج حالات الاستخدام. يحدد فريق التطوير شكل قالب وثيقة حالات الاستخدام. يتكون القالب غالباً من عدة أجزاء بالإضافة إلى رأس الوثيقة Header الذى يتضمن معلومات عن مؤلف الوثيقة وتاريخ إنشائها وغيرها. يوضح شكل (٥-٧) أحد أشكال القوالب المستخدمة لوثيقة حالات الاستخدام. تبدأ الوثيقة بعنوان لحالة الاستخدام يليه وصف مختصر للحالة، ثم عرض لمجموعة الفاعلين للحالة. تحتوى الوثيقة أيضاً على وصف لأى شروط مسبقة يجب أن تتوافر قبل بدء حالة الاستخدام وكذلك وصف لأى شروط لاحقة يجب تنفيذها بعد انتهاء تنفيذ حالة الاستخدام. يحتوى الجزء الأساسى لحالة الاستخدام على مجموعة الخطوات الأساسية لتنفيذ الحالة وتسمى «التدفق الأساسى» وهو يمكن أن يحتوى على مجموعة من التدفقات الفرعية

بالإضافة إلى تدفقات استثنائية تصف مجموعة الأفعال التي يجب اتخاذها عند حدوث أخطاء فى أثناء التنفيذ.

وفيما يلى عرض لمحتوى كل جزء من أجزاء وثيقة حالات الاستخدام لتطبيق إدارة البريد الإلكترونى EM.

وصف مختصر لحالة الاستخدام:

تهدف حالة الاستخدام إلى عرض محتويات الرسائل الموجودة بقاعدة البيانات وإرسالها إلى العملاء المقيدين بالنظام.

الفاعلون:

موظف إدارة العملاء.

قاعدة بيانات الرسائل والعملاء.

جهة اتصال.

شروط مسبقة:

تخزين الرسائل المطلوب إرسالها فى قاعدة البيانات.

منح الموظف المختص بإدارة العملاء صلاحية استخدام قاعدة البيانات.

شروط لاحقة:

التعديل فى معلومات قاعدة البيانات لتعكس حالة الرسائل التى تم إرسالها بنجاح.

الاستمرار فى الاتصال بقاعدة البيانات حتى يتم إرسال باقى الرسائل.

التدفق الأساسى:

يطلب النظام معلومات المستخدم وكلمة المرور.

بعد نجاح عملية الدخول، يستعرض النظام قائمة الاختيارات المتاحة للمستخدم وهى:

١- استعراض محتويات الرسالة.

٢- عرض قائمة الرسائل التى لم ترسل.

٣- إرسال رسالة.

٤- خروج.

رقم النسخة: المؤلف:	تاريخ الإنشاء:
عنوان حالة الاستخدام Use Case Title وصف مختصر لحالة الاستخدام Brief Description: الفاعلون Actors: شروط مسبقة Preconditions: شروط لاحقة Postconditions: التدفق الأساسى Basic Flow: التدفقات الفرعية Subflows: تدفقات استثنائية Exception Flows:	

شكل (٥-٧) أحد قوالب وثيقة حالات الاستخدام

التدفقات الفرعية:

يحتوى التدفق الأساسى على مجموعة التدفقات الفرعية التالية:

S١: استعراض محتويات الرسالة:

يقوم الموظف بإدخال رقم الرسالة المطلوبة.

يتم عرض محتوى الرسالة.

يتم عرض قائمة الاختيارات مرة أخرى.

S٢: عرض قائمة الرسائل التى لم ترسل:

عرض رسالة تحتوى على عدد الرسائل التى لم ترسل بعد.

عرض المحتويات العامة عن كل رسالة (رقمها، التاريخ، الموضوع، العميل).

يتم عرض قائمة الاختيارات مرة أخرى.

S٣: إرسال الرسالة.

يقوم الموظف بإدخال رقم الرسالة المطلوب إرسالها.

تنفيذ عملية إرسال الرسالة.

تعديل بيانات الرسائل فى قاعدة البيانات.

يتم عرض قائمة الاختيارات مرة أخرى.

تدفقات استثنائية:

يحتوى النظام على التدفقات الاستثنائية التالية:

E١: اسم مستخدم أو كلمة مرور غير صحيحة:

هى رسالة تظهر للمستخدم عند إدخال اسم مستخدم User Name أو كلمة مرور Password غير صحيحة.

يسـمح النظام بأن يقوم المسـتخدم بإعادة إدخال بيانات المستخدم وكلمة المرور أو الخروج.

يسمح النظام بثلاث محاولات فقط لإدخال البيانات بشكل صحيح.

E٢: اختيار غير صحيح:

هى رسـالة تظهر للمسـتخدم عند إدخال رقـم لا يتوافق مع أرقـام الاختيارات المعروضة فى التدفق الأساسى للنظام.

يتم عرض قائمة الاختيارات مرة أخرى.

E٣: عدد كبير جداً من الرسائل:

هى رسالة يمكن أن تظهر فى أثناء تنفيذ التدفق الفرعى S١ (عرض قائمة الرسائل التى لم ترسل) إذا كان عدد الرسائل غير المرسلة كبيراً يتعدى الرقم المحدد من قبل لعدد الرسائل التى يمكن عرضها.

يقوم النظام بعرض عدد الرسائل المسموح بها.

يتم عرض رسالة تفيد بوجود رسائل أخرى لم ترسل وموجودة بقاعدة البيانات.

E٤: رسالة لم يتم إرسالها:

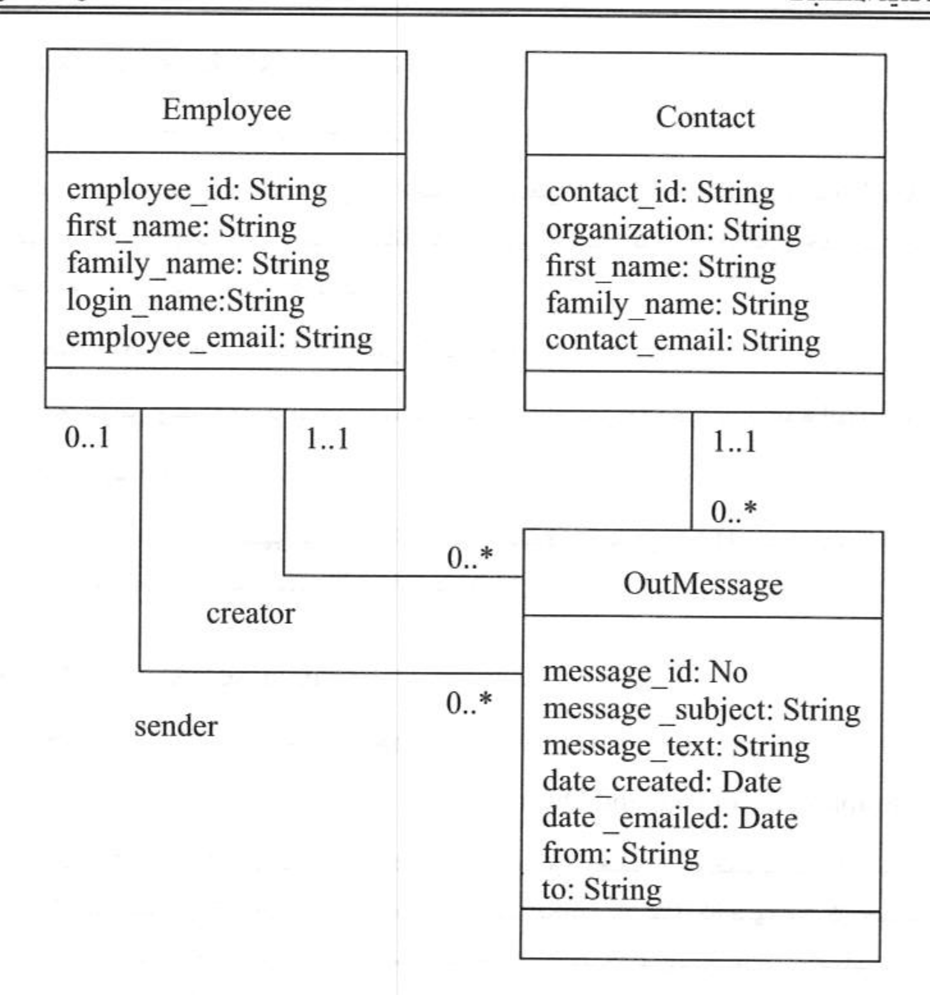
هى رسالة يمكن ظهورها فى أثناء تنفيذ التدفق الفرعى S٣ (إرسال رسالة) فى حالة عدم إتمام عملية إرسال الرسالة بنجاح.

٥-٣-١٣ الأصناف Classes:

يحتوى نظام إدارة البريد الإلكترونى على ثلاثة أصناف هى:

Employee، Contact، and Outmessage

يوضح شكل (٥-٨) مخطط الأصناف للنظام. يحتوى المخطط على الأصناف والعلاقات بينها مع توضيح اسم العلاقة وعدد الكائنات المشاركة فيها من كل صنف. وقد تم عرض مجموعة الخصائص المهمة لكل صنف ولم يتم عرض العمليات فى هذه المرحلة الأولية لنموذج تحليل المتطلبات. نلاحظ أيضاً من المخطط وجود أكثر من علاقة ربط بين الصنفين Employee، Outmessage. العلاقة الأولى تسمى «Creator» وهى تصف العلاقة بين الرسالة والموظف الذى قام بإنشائها وتخزينها فى قاعدة البيانات. أما العلاقة الثانية فتسمى «Sender»، وهى تصف العلاقة بين الرسالة والموظف الذى سيقوم بإرسالها.



شكل (٥-٨) مخطط الأصناف للتطبيق EM

الفصل السادس
التصميم المعمارى للنظام
Architectural System Design

يعتبر التصميم المعمارى للنظام البرمجى هو الأساس الذى تبنى عليه جميع الحلول والتصميمات الأخرى داخل النظام. ولذلك فإنه لا يمكن بناء أى نظام برمجيات بدون تصميم معمارى مسبق للنظام. يهتم التصميم المعمارى للبرمجيات بعمليات تنظيم الوحدات البرمجية مثل: الأصناف Classes، والمكونات Components، والحزم Packages وكذلك توضيح الاتصالات الداخلية بين الوحدات.

٦-١ الطبقات المعمارية وإدارة التبعيات Architectural Layers and Dependency Management:

يتميز التصميم المعمارى للنظم البرمجية باعتماده أسلوب هيكلة النظام فى شكل طبقات من الوحدات مع تنظيم الاتصالات بين الوحدات. يؤدى هذا الأسلوب إلى تقليل درجة التعقيد فى الهيكل المعمارى للنظام وتحسين درجة الفهم للتبعيات بين الوحدات. ومن الفوائد المهمة أيضاً لاستخدام التصميم المعمارى ضمان استخدام أسلوب الوحدات البرمجية المستقلة والتفاعل فيما بينها وهو يتوافق تماماً مع أسلوب الكائنات الموجهة Object-Orientation لإنتاج البرمجيات.

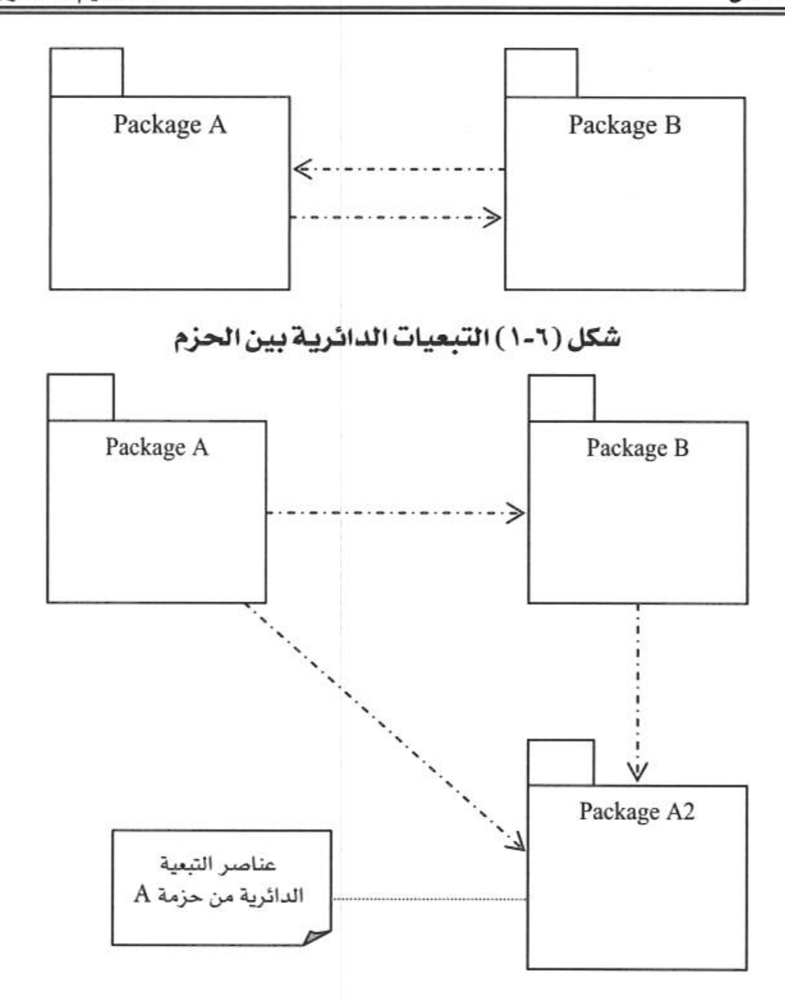
٦-١-١ الوحدات البرمجية المعمارية Architectural Modules:

يعتبر الصنف Class من الوحدات البرمجية التى تستخدم على نطاق واسع خلال مراحل التطوير المختلفة لمشروع البرمجيات، فهو يستخدم لوصف مجموعة الفاعلين Actors المتعاملين مع النظام وكذلك مجموعة الكائنات البرمجية للنظام البرمجى مثل: كائنات التعامل مع قواعد البيانات، وكائنات واجهات المستخدم، وكائنات إجراء العمليات، وغيرها. ويستخدم الصنف فى المستويات المختلفة مثل: الوحدة البرمجية أو التطبيق أو النظام العام. ويستخدم الصنف أيضاً فى جميع مراحل التطوير مثل:

التحليل والتصميم والبرمجة. يتم تجميع الأصناف حسب الإطار المعمارى المناسب لتطوير المشروع فى شكل مجموعة من الحزم Packages يتم تنظيمها أيضاً فى شكل معمارى يصف العلاقات بينها. يسـهم التصميم المعمارى فى تنظيم عملية التبعيات بين الوحدات، فيمكـن اعتبار إحدى الوحدات A معتمدة على وحدة أخرى B إذا كان التغيير فـى الوحدة B يؤدى إلى تغيير فـى الوحدة A. وكما ذكرنا سـابقاً فإن لغة النمذجة الموحدة UML تدعم أسـلوب تصميم الحزم التى يتم تطبيقها بشكل مماثل فـى أثناء البرمجة باسـتخدام لغات البرمجة الحديثة المبنية على أسـلوب الكائنات الموجهـة مثل لغة جافا Java. تحتوى الحزمة علـى مجموعة من الأصناف التى يتم استدعاؤها مرة واحدة عند استدعاء الحزمة، وكذلك يمكن حذفها كاملة من التطبيق. يمكن أن تحتوى الحزمة على حزم أخرى بداخلها. وعلى هذا فإن أى صنف يمكن أن يكون عضواً فى حزمة واحدة فقط ولكن يمكن تصديره إلى حزم أخرى.

٦-١-٢ تبعيات الحزم البرمجية Packages Dependencies:

تحتاج عملية تنظيم الحزم بشـكل معمارى إلـى إدارة لعملية التبعيات بين الحزم. ومن المعروف أنه لزيادة كفاءة عملية إعادة اسـتخدام الوحدات البرمجية، فإنه يجب علينا أن نقلل من الاعتمادية بين الوحدات بقدر الإمكان وكذلك التخلص من التبعيات غيـر الضروريـة بين الوحدات من التصميـم المعمارى للنظام. ومـن التبعيات غير المرغوب فيها وتسـبب مشكلات كبيرة فى أثناء التطبيق التبعيات الدائرية أو المغلقة Circular Dependencies. يوضح شكل (٦-١) مثالاً لتبعية دائرية بين حزمتين، حيث تعتمـد الحزمة A على الحزمـة B وكذلك تعتمد الحزمة B على الحزمة A. ويؤدى ذلك إلى مشكلات كبيرة فى أثناء التطبيق والصيانة وإعادة الاستخدام. يمكن التغلب على تلك المشـكلة بإضافة حزمة ثالثة A٢ تحتوى على عناصر من الحزمة A التى تعتمد عليها الحزمة B، كما هو موضح بشـكل (٦-٢). وعلى هذا فلم تعد الحزمة B تعتمد على الحزمة A ولكنها أصبحت تعتمد على الحزمة A٢ فقط. وبالطبع إن الحزمة A تعتمد على الحزمة A٢، وبذلك تم إزالة التبعية الدائرية.



شكل (٦-١) التبعيات الدائرية بين الحزم

شكل (٦-٢) إزالة التبعيات الدائرية

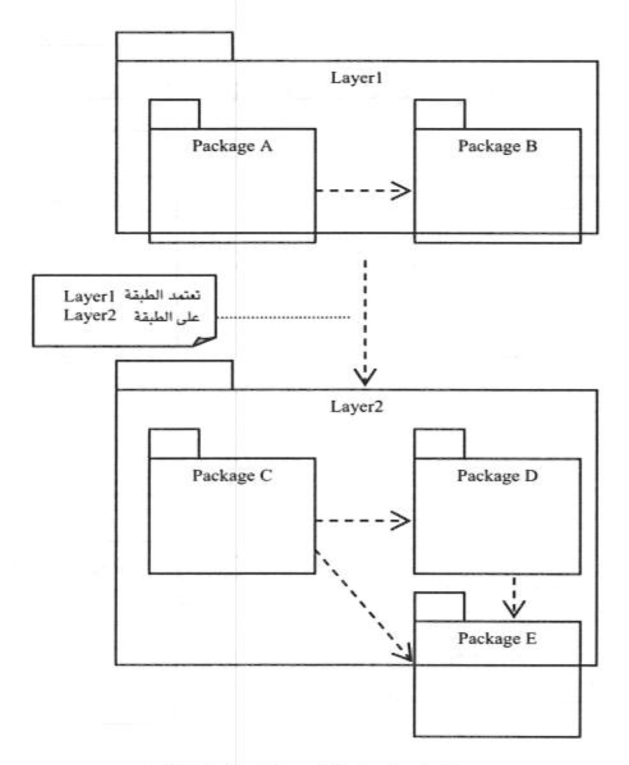
٦-١-٣ تبعيات الطبقة Layer Dependencies:

يمكننا تجميع الحزم وتنظيمها فى شكل طبقات هرمية Hierarchical Layers وذلك فى محاولة لدعم عملية وصف معمارية النظام. ومن المعروف أن الطبقات تأخذ الشكل الرأسى، إذ تحتوى كل طبقة على مجموعة من الحزم مما يؤدى إلى التصميم الهرمى للحزم كما هو موضح بالشكل (٦-٣). يهدف التصميم الجيد لمعمارية الطبقات إلى ثلاثة أهداف هى:

١- أن تساعد هرمية الطبقات على التكامل مع باقى مكونات النظام.

٢- أن تؤدى هرمية الطبقات إلى تقليل التبعيات بين الحزم.

٣- أن تسهم هرمية الطبقات فى إمكانية الحصول على إطار عمل مستقر لعملية تطوير النظام.



شكل (٦-٣) تبعيات الطبقة

٦-١-٤ تبعيات الصنف Class Dependencies:

يجب التخلص من التبعيات الدائرية الموجودة بين الطبقات وذلك لضمان الحصول على تصميم معمارى مستقر للنظام. ولإمكانية تحقيق ذلك يجب معرفة الأسباب التى تؤدى إلى وجود التبعيات الدائرية. فمن المعروف أن التبعيات بين الطبقات تنشأ نتيجة وجود تبعيات بين الحزم فى الطبقات، وأن التبعيات بين الحزم تنشأ نتيجة وجود تبعيات بين الأصناف داخل الحزم. يوضح شكل (٦-٤) هذا المفهوم، إذ نجد أن الطبقة Layer١ تعتمد على الطبقة

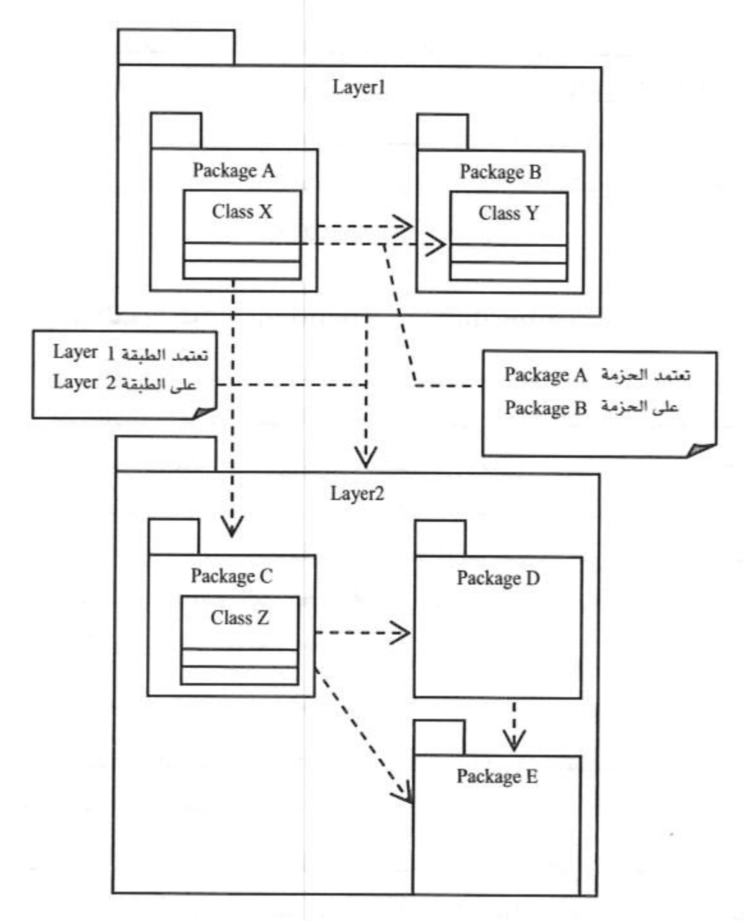
Layer٢ لأن هناك صنف Class X فى الطبقة Layer١ يعتمد على الصنف Class Z فى الطبقة Layer٢. ونجد أيضاً أن الحزمة Package A تعتمد على الحزمة Package B لأن هناك صنف Class X فى الحزمة Package A يعتمد على الصنف Class Y فى الحزمة Package B. وعلى هذا فانه يجب التركيز على إزالة التبعيات الدائرية التى يمكن أن تنشأ بين الأصناف وذلك يؤدى إلى إزالة أى تبعيات دائرية بين الحزم أو بين الطبقات.

٦-١-٥ تبعيات التوريث Inheritance Dependencies:

تعتبر الاعتمادية بين الأصناف الناتجة عن خاصية التوريث من التبعيات التى يمكن أن تؤدى إلى مشكلات كبيرة. فمن المعروف أنه فى حالة تطبيق أسلوب التوريث نجد أن الصنف الفرعى sub class يرث كل خصائص وطرق الصنف العلوى super class مع إمكانية قيامه بتغيير أو استبدال بعض هذه الخصائص والطرق وإضافة خصائص وطرق جديدة له. ويؤدى ذلك إلى تغيير فى سلوك النظام فى أثناء التشغيل اعتماداً على سلوك الكائنات الفرعية المستخدمة فى أثناء تنفيذ عملية التوريث فعلياً. يمكننا تصنيف تبعيات التوريث إلى نوعين هما:

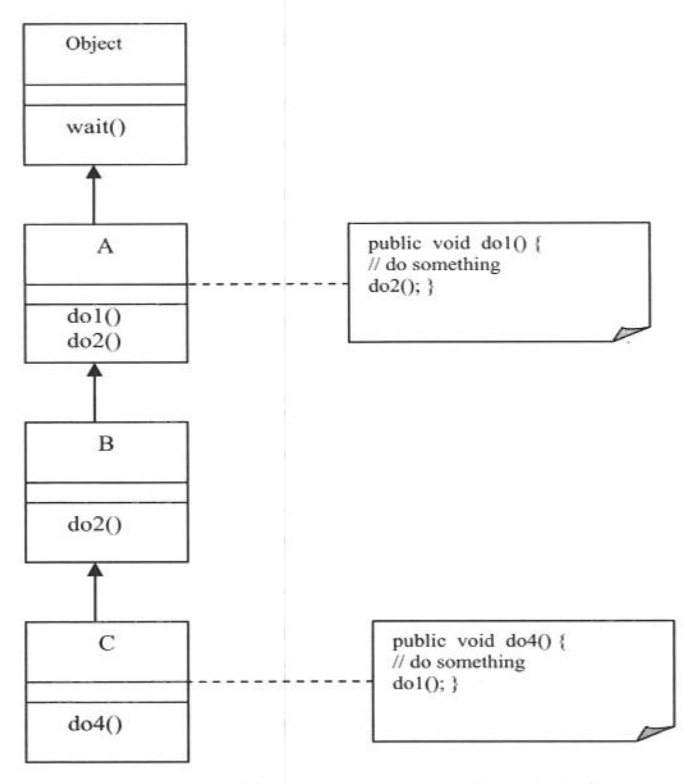
- تبعيات بين الأصناف فى شجرة توريث فى أثناء وقت الترجمة compile time.
- تبعيات بين الكائنات فى وقت التنفيذ run time وهى تستخدم خدمات تقدمها أصناف موجودة فى شجرة توريث.

يوضح شكل (٦-٥) مثالاً على التبعيات بين الأصناف فى شجرة توريث فى أثناء وقت الترجمة. تؤدى علاقة التعميم generalization relationship إلى وجود تبعيات فى اتجاه الأسهم، حيث نجد أن الصنف C يعتمد على الصنف B وهو يعتمد بدوره على الصنف A الذى يعتمد على الصنف العام object. نلاحظ أيضاً من الشكل وجود تبعيات فى وقت التنفيذ من خلال الطريقة الموجودة فى الصنف B الذى يمكن أن يستبدل الطريقة نفسها do٢() الموجودة فى الصنف A. ويمكن أيضاً أن تنشأ تبعية دائرية نتيجة للتوريث بين الأصناف. ففى حالة قيام الطريقة do١() فى الصنف A باستدعاء الطريقة do٢() فى الصنف نفسه ويمكن استبدالها بالطريقة do٢() الموجودة بالصنف B، فإن ذلك يؤدى إلى وجود تبعية دائرية يصعب التحكم فيها وتؤدى إلى مشكلات كبيرة.

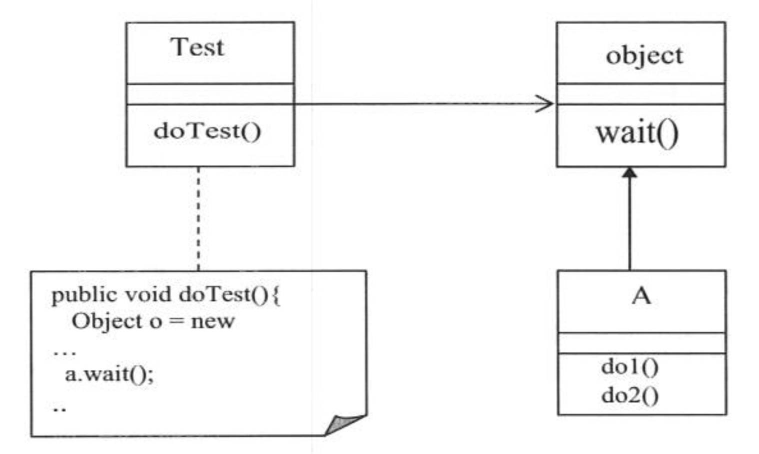


شكل (٦-٤) تبعيات الصنف

يوضح شكل (٦-٦) مثالاً على تبعيات التوريث فى أثناء التنفيذ run time. نلاحظ من المثال أن التبعية تحدث نتيجة التوريث عندما يستدعى الكائن test الطريقة wait() من خلال الكائن A فى وقت التنفيذ، إذ إن الكائن A يرتبط بالكائن object بالتوريث ويمكنه أن يغير من سلوك الطريقة wait() فى أثناء التنفيذ. ويتطلب ذلك الحرص الشديد من فريق التصميم ومراعاة تجنب حدوث تلك الأنواع من التبعيات الدائرية، بالإضافة إلى الحرص عند استخدام أسلوب الاستبدال للطرق method overriding فى وجود التبعيات بين الأصناف.



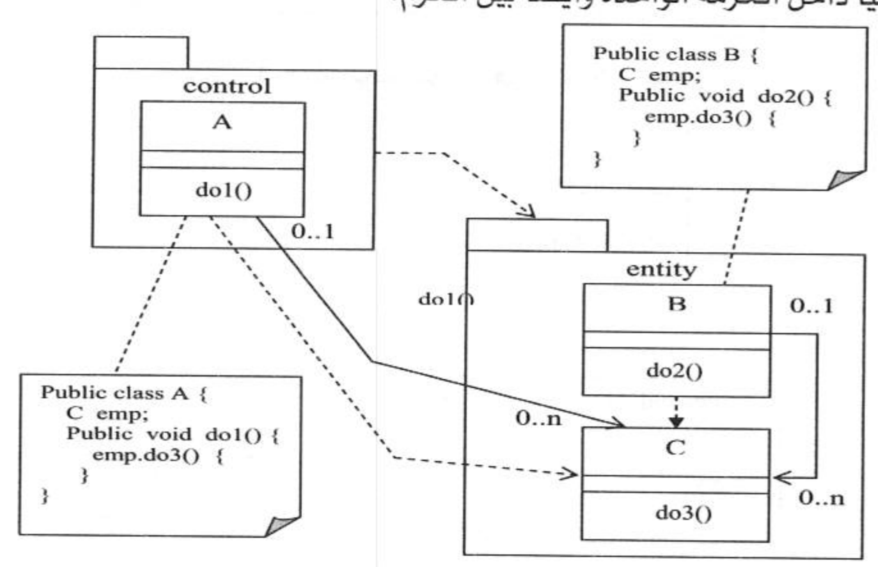
شكل (٦-٥) تبعيات التوريث فى أثناء الترجمة



شكل (٦-٦) تبعيات التوريث فى أثناء التنفيذ

٦-١-٦ تبعيات الطرق Method Dependencies:

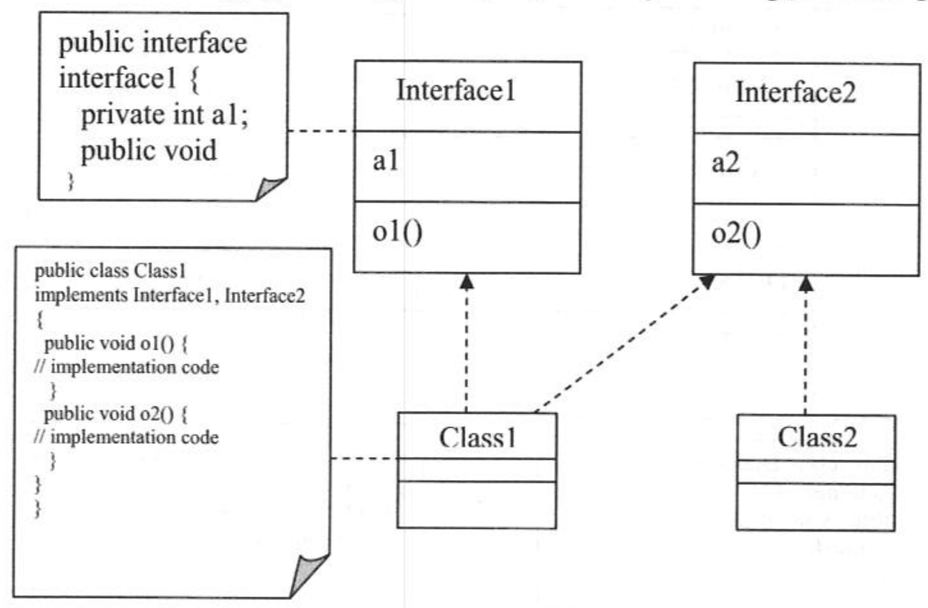
كما ذكرنا سابقاً فإن التبعيات بين الأصناف تكون نتيجة مباشرة للتبعيات بين الطرق Methods الموجودة داخل الأصناف. يوضح شكل (٦-٧) مثالاً لتبعيات الطرق وكيفية تسببها فى وجود تبعيات بين الأصناف وكذلك تبعيات بين الحزم. نلاحظ من المثال أن الصنف A يحتوى على الطريقة do1() التى تستدعى الطريقة do3() الموجودة فى الصنف C مما يؤدى إلى تبعية بين الصنفين A،C ومن ثم يؤدى إلى تبعية بين الحزمتين التى ينتمى إليها كل صنف. ونلاحظ أيضاً أن الصنف B يحتوى على الطريقة do2() التى تستدعى أيضاً الطريقة do3() الموجودة بالصنف C ومن ثم يؤدى إلى تبعية بين الصنفين B،C. يتضمن المثال أيضاً علاقة الترابط بين الأصناف Association بشكل مباشر وهو يساعد كثيراً فى توضيح التبعيات الموجودة بين الأصناف والتى يصعب اكتشافها من خلال تحليل أكواد البرنامج. وبالطبع فإنه لا يمكن التخلص من جميع أنواع التبعيات الموجودة بين الأصناف؛ لأنها تأتى نتيجة أسلوب التخاطب الطبيعى بين كائنات الأصناف عن طريق استدعاء الطرق الموجودة بها. ولكن يجب حصر تلك التبعيات داخل الحزمة الواحدة وتجنب حدوثها بين الأصناف الموجودة فى حزم مختلفة. بالاضافة إلى تجنب حدوث التبعيات الدائرية نهائياً داخل الحزمة الواحدة وأيضاً بين الحزم.



شكل (٦-٧) تبعيات الطرق

٦-١-٧ الواجهات Interfaces:

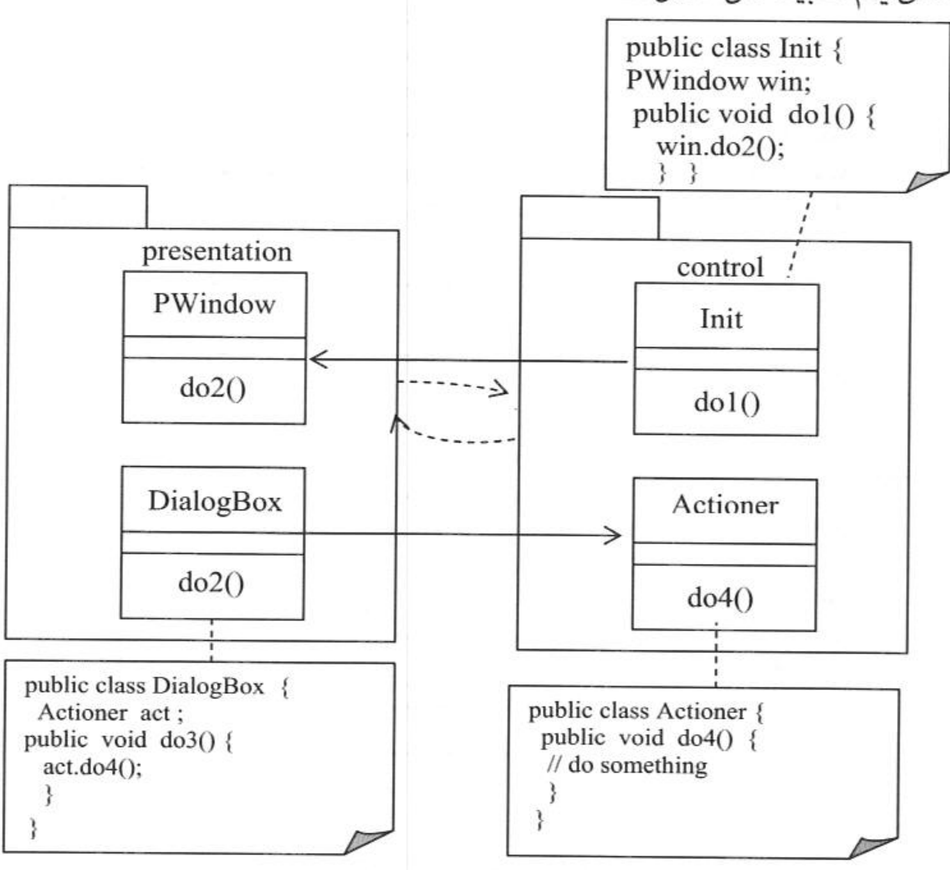
كما ذكرنا سابقاً فإن الواجهة هى: مجرد تعريف لمجموعة من الخصائص والطرق التــى لا يمكن إنشــاء كائنات منها بطريقة مباشــرة ولكن تحتاج إلــى صنف آخر يقوم بتطبيقها أو تحقيقها وذلك مــن خلال توصيف الأكواد التفصيلية للطرق الموجودة فى صنــف الواجهة أى أنها تقوم بتحقيقهــا فعلياً. يعتبر ذلك نوعاً من أنواع التبعيات التى تسمى تبعية التطبيق Implementation Dependency وهى توجد بين صنف الواجهة والصنف الذى يقوم بتطبيقه. يمكن أن يقوم صنف واحد بتحقيق أكثر من صنف واجهة، ويمكــن أن يقــوم أكثر من صنف واحــد بتحقيق صنف الواجهة. يوضح شــكل (٦-٨) مثــالاً على تبعية التطبيق من خلال علاقة التبعية بين صنف الواجهة وصنف التحقيق. نلاحظ من الشــكل أن اتجــاه العلاقة يكون ناحية صنف الواجهــة. ونلاحظ أيضاً أن الصنــف Class1 يقوم بتحقيق صنفين الواجهة Interface2، وInterface1 وكذلك نجد أن كلا الصنفين Class1، و Class2 يمكنهما تحقيق صنف الواجهة Interface2.



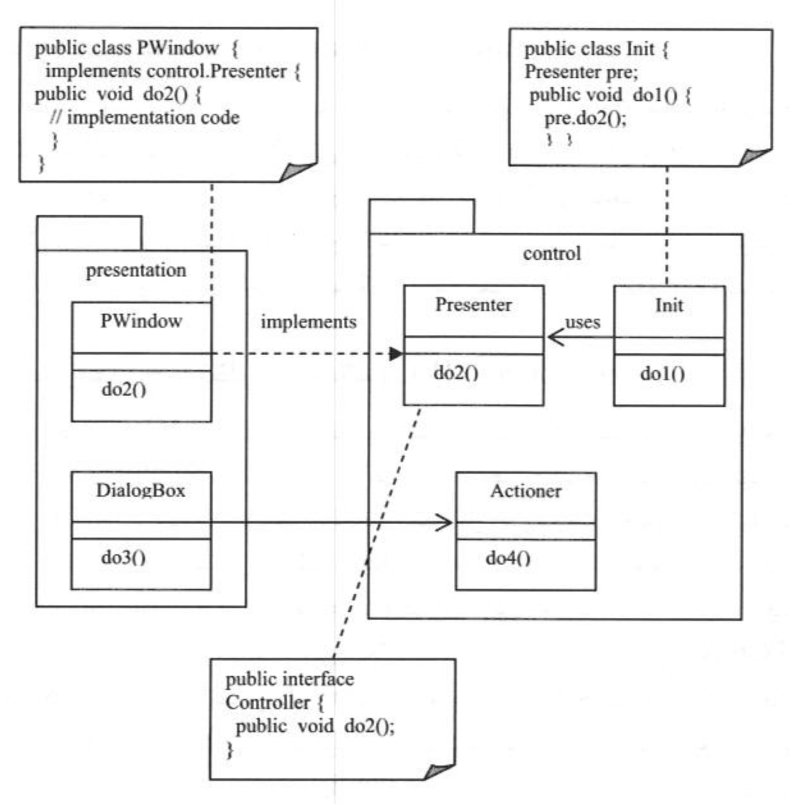
شكل (٦-٨) تبعية التطبيق

ومع ذلك فإن من مميزات استخدام صنف الواجهة هو إزالة التبعية الدائرية Circular Dependency بين الأصناف. ولتوضيح ذلك نستعين بالمثال الموضح بالشكل (٦-٩) وهو يحتوى على تبعية دائرية بين الحزمتين Presentation و Control.

ولحل هذه المشكلة وكسر التبعية الدائرية يتم استخدام صنف الواجهة فى إحدى الحـزم كما هو موضح بالشـكل (٦-١٠). يقوم صنف الواجهـة Presenter بتعريف الطريقـة ()do2 التى يسـتخدمها الصنـف Init والموجود معه فى الحزمة نفسـها Control ويتـم تطبيق صنـف الواجهة من خلال الصنـف PWindow الموجود فى الحزمة Presentation. وعلى هذا فإن الصنف Init سـوف يستخدم صنف الواجهة الذى يتم تطبيقه من خلال الصنف PWindow.



شكل (٦-٩) مثال على التبعية الدائرية



شكل (٦-١٠) استخدام صنف الواجهة لحل التبعية الدائرية

٦-١-٨ معالجة الأحداث Event Processing:

تنتج التبعيات بين الطرق Method Dependencies نتيجة للاتصالات التى تتم بين الكائنات فى شكل رسائل من الكائن المرسل إلى كائن آخر يطلب منه تنفيذ خدمة معينة. فعندما يرسل كائن A رسالة إلى كائن B يطلب منه تنفيذ خدمة معينة فإنه توجد تبعية بين الكائنين، إذ يعتمد الكائن A على الكائن B؛ لأنه ينتظر منه معلومات هى ناتج تنفيذ طرق موجودة لدى الكائن B. يمكن التعبير عن آلية تلك الاتصال من خلال كائن يمثل أصل الحدث يسمى Publisher Object ومجموعة من الكائنات التى تستجيب للحدث وتؤدى مجموعة الأفعال وتسمى Subscriber Objects. وأحياناً يتم استخدام كائن منفصل يسمى Registrator Object لتنفيذ عملية الاتصال بين الكائن Publisher Object والكائن Subscriber Object.

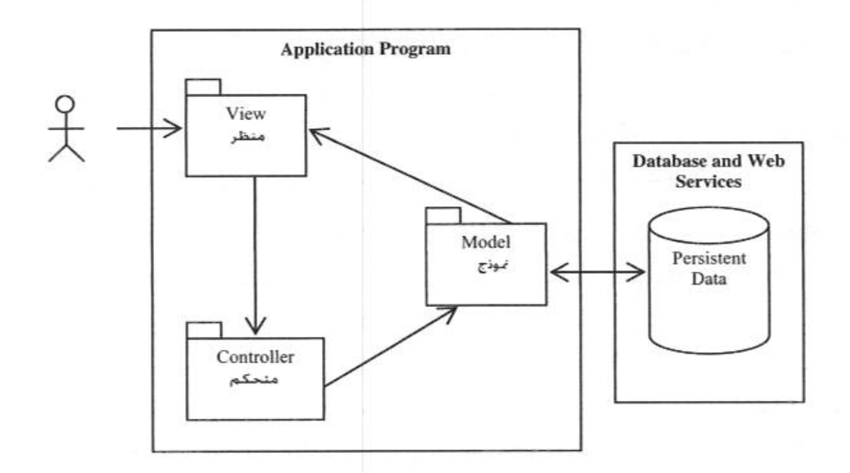
٦-٢ أطر معمارية Architectural Frameworks:

تهتم النظم المعتمدة على أسلوب الكائنات الموجهة ببناء أطر معمارية تستخدم فى أثناء عمليات التصميم والبرمجة التى تعزز عملية إعادة الاستخدام Reuse فى مرحلتى التصميم والبرمجة. سنقدم الآن عرضاً لبعض تلك الأطر المعمارية التى تدعم عملية إعادة الاستخدام، إذ يقوم المصمم والمبرمج بتخصيص هذه الأطر والإضافة إليها لتناسب احتياجاته وذلك فى حدود الهيكل التصميمى للإطار المعمارى المستخدم.

٦-٢-١ إطار نموذج- منظر - متحكم Model-View-Controller (MVC):

يعتبر إطار MVC أحد أطر التصميم المبنية على الكائنات وقدمت من خلال بيئة التطوير Smalltalk-80. يقوم هذا الإطار على فكرة الفصل بين المفاهيم للتصميم المبنى على الكائنات وذلك بتقسيم كائنات التطبيق إلى ثلاث مجموعات ترتبط بثلاثة أصناف مجردة فى شكل الإطار Controller و View و Model من خلال علاقة التعميم Generalization، إذ ترث منها وتضيف إليها خصائصها المميزة. يتم تمثيل كائنات البيانات Data Objects من خلال كائنات النموذج للإطار Model Objects. يتم إشعار كل من كائنات المنظر View Objects وكائنات المتحكم Controller Objects بأى تغييرات تحدث فى كائنات النموذج باستخدام أسلوب معالجة الأحداث، إذ يعتبر النموذج هو مؤسس الحدث Event Publisher وتستجيب كل من كائنات المنظر والمتحكم لهذا الحدث. يتم تمثيل كائنات واجهة المستخدم GUI Objects من خلال كائنات المنظر للإطار View Objects وهو يصف حالة النموذج بشكل رسومى يناسب المستخدم. يستجيب المنظر للتغييرات التى تحدث فى كائنات النموذج ويقوم بتحديث البيانات التى تعرضها للمستخدم. ومن الطبيعى أن يرافق كل كائن منظر كائن آخر متحكم لتمثيل الأحداث.

يتم تمثيل الأحداث من خلال كائنات المتحكم التى تستجيب لطلبات المستخدم فى أثناء التفاعل مع النظام من خلال كائنات المنظر. وتعتبر كائنات المتحكم بمنزلة الوسيط بين كائنات النموذج وكائنات المنظر، إذ تقوم بتفسير الحدث مثل النقر على الفأرة أو مفاتيح الإدخال وتحويلها إلى أفعال تطبق على كائنات النموذج. يوضح شكل (٦-١١) إطار MVC والاتصال بين الكائنات الثلاثة، إذ يقوم كائن المنظر باستقبال الحدث من مستخدم النظام ويقوم بتمريره إلى الكائن المتحكم لتفسيره وتحويله إلى الفعل المناسب ليتم تطبيقه على كائنات النموذج.



شكل (٦-١١) إطار MVC

يحقق استخدام هذا الإطار الثلاثى العديد من المميزات مثل:

- السماح بالتطوير المنفصل لكل من واجهة المستخدم والنموذج الذى يتضمن كلاً من بيانات وعمليات كائنات نظام الأعمال.
- إمكانية استخدام أكثر من واجهة مستخدم بدون إجراء تعديلات على النموذج.
- إمكانية إعادة تصميم النموذج مع الحفاظ على واجهة المستخدم بدون تغيير.
- إمكانية تغيير طريقة استجابة واجهة المستخدم للأحداث بدون تغيير واجهة المستخدم.

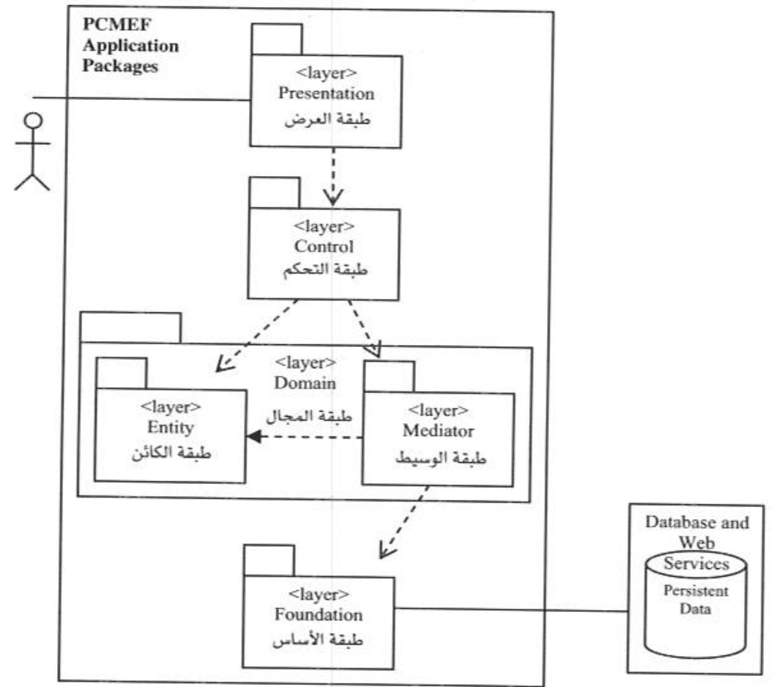
٦-٢-٢ إطار عرض - تحكم - وسيط - كائن - أساس Presentation-Control-Mediator-Entity-Foundation (PCMEF):

نقدم الآن إطار عمل آخر يسمى PCMEF وهو يتكون من مجموعة من الطبقات ويأخذ الشكل المعمارى الهرمى. يتكون الإطار من أربع طبقات رأسية هى:

- العرض Presentation.
- التحكم Control.
- المجال Domain.
- الأساس Foundation.

وتتكون طبقة المجال من حزمتين هما الكائن Entity والوسيط Mediator. يوضح شكل (٦-١٢) التنظيم الهرمى لطبقات الإطار. نلاحظ من الشكل وجود تبعيات بين الطبقات من أعلى إلى أسفل، حيث تعتمد طبقة العرض على طبقة التحكم التى تعتمد بدورها على طبقة المجال، وتعتمد حزمة الوسيط على حزمة الكائن وعلى طبقة الأساس.

تحتوى طبقة العرض Presentation على كائنات واجهة المستخدم GUI Objects تستخدم فى تعامل المستخدم مع النظام. وتحتوى طبقة التحكم Control على الكائنات المسؤولة عن معالجة تفاعلات المستخدم مع النظام من خلال مجموعة الخدمات الوظيفية التى يقدمها النظام. وتحتوى حزمة الكائن Entity على كائنات النظام التى سيتم تخزينها فى قاعدة البيانات أو التى يتم استرجاعها من قاعدة البيانات. تقوم الحزمة Mediator بدور الوسيط أو قناة اتصال بين كائنات حزمة الكائن Entity وكائنات طبقة الأساس Foundation. تقوم طبقة الأساس بجميع العمليات المسؤولة عن الاتصال بقواعد البيانات وإدارة العمليات التى تتم على البيانات مثل: التسجيل والاستعلام والاسترجاع وغيرها.



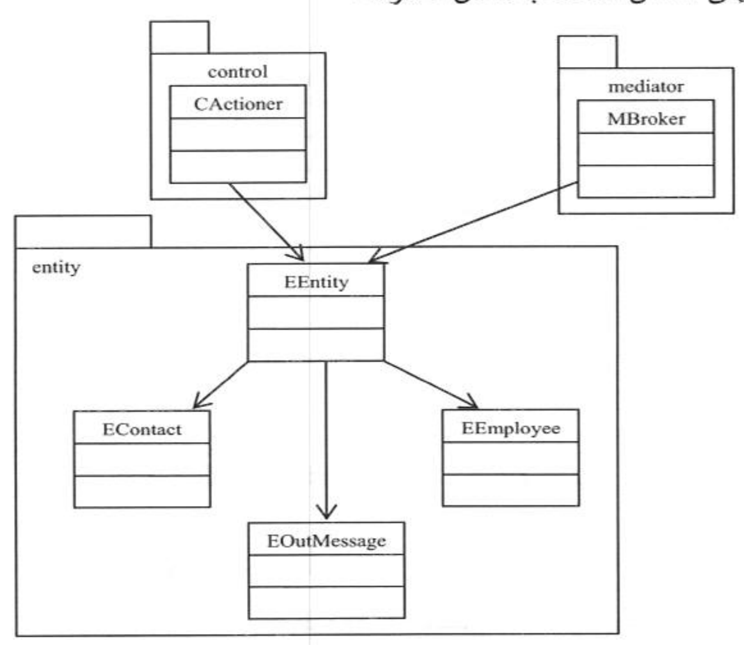
شكل (٦-١٢) إطار PCMEF

٦-٣ أنماط معمارية Architectural Patterns:

نستعرض فى هذا الجزء بعض الأنماط المعمارية التى تم استخدامها فى تصميم الأطـر المعمارية مثل إطار PCMEF. تسـتخدم هذه الأنماط لحل بعض مشـكلات التصميم مثل: التحكم فى التبعيات وإعادة اسـتخدام الوظائف واسـتخدام أساليب التجريد Abstraction وإخفاء المعلومات Information Hiding (Gamma et al, 1995).

٦-٣-١ الواجهة Facade:

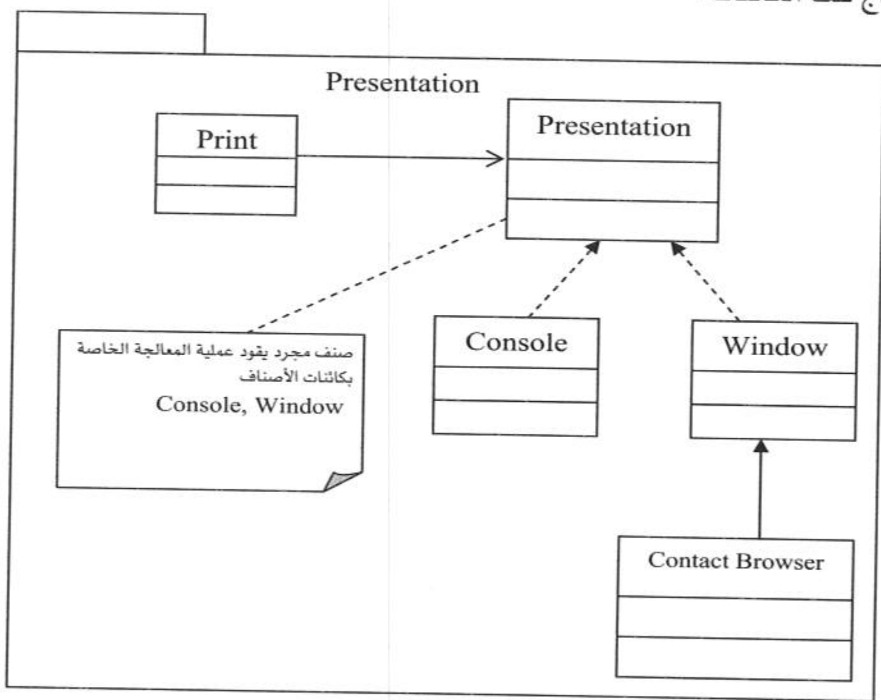
يعتبر الغرض الأساسـى من استخدام نمط الواجهة هو تقليل عدد الاتصالات بين الحزم لأقل ما يمكن مما يقلل من التبعيات بين الحزم. يتم اسـتخدام صنف الواجهة فى المسـتوى الأعلى داخل الحزمة ويكون هو نقطة الدخول الرئيسـية للحزمة عند التعامل مع الحزم الأخرى. تقوم الواجهة بتوجيه طلب الخدمة إلى الكائن المناسـب لتنفيذها داخل الحزمة مما يؤدى إلى تقليل عدد مسارات الاتصال بين كائنات الحزمة والحزم الأخرى. يوضح شكل (٦-١٣) كيفية اتصال كل من حزمة Mediator وحزمة Control مع حزمة Entity من خلال صنف الواجهة EEntity الذى يقوم بعملية توجيه الاتصال إلى الكائن المناسب داخل الحزمة.



شكل (٦-١٣) نمط الواجهة

٦-٣-٢ المصنع المجرد Abstract Factory:

يقدم هذا النمط أسـلوباً آخر لتقليل التبعيات بين الحزم من خلال واجهة مجردة تقوم بإنتاج مجموعة من الكائنات المختفية خلف الواجهة. يؤدى هذا الأسـلوب إلى إمكانيـة أن يتصرف التطبيق بأسـلوب مختلف عندما يتعامل مـع الكائنات المولدة حسـب نوع الخدمة المطلوبة. يوضح شـكل (٦-١٤) مثالاً لاستخدام نمط المصنع المجـرد، حيث يمكن للتطبيق اسـتخدام كائنات لوحة المفاتيح PConsole أو كائنات النوافـذ PWindow وذلك من خلال الصنف المجـرد IPPresentation الذى يقوم بإنتاج تلك الكائنات.

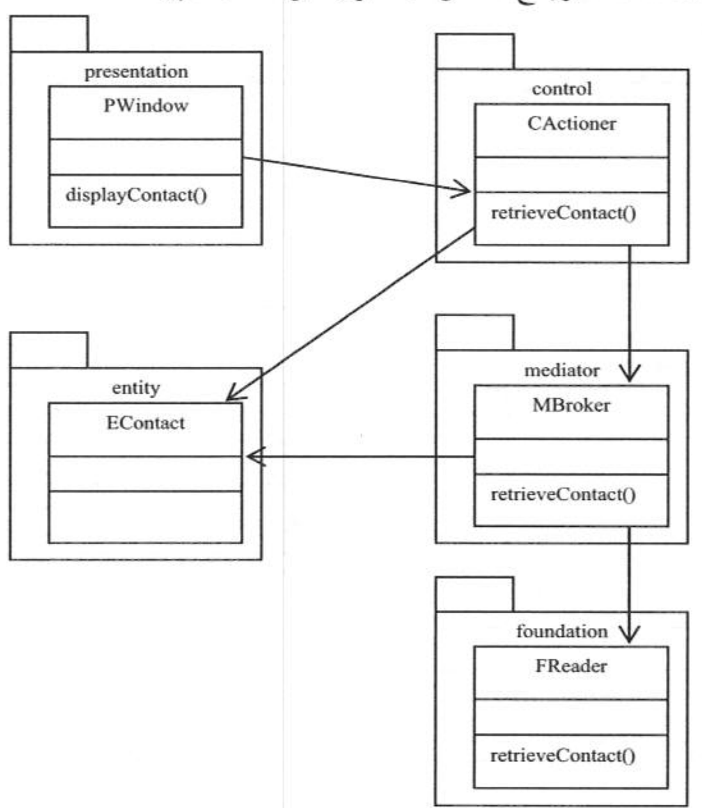


شكل (٦-١٤) نمط المصنع المجرد

٦-٣-٣ سلسلة المسؤوليات Chain of Responsibilities:

يعتبر الغرض من اسـتخدام نمط سلسلة المسؤوليات هو تجنب الاقتران المباشر بين الكائن المرسـل لطلب الخدمة مع الكائن المسـتقبل لهـا وإعطاء الفرصة لأكثر

من كائن للتعامل مع الطلب. يوضح شــكل (٦-١٥) مثالاً لاستخدام هذا النمط، حيث يقوم الكائن PWindow بطلب خدمة () display Contact بأن يرســل رسالة للكائن CActioner الذى يمرر بدوره هذا الطلب إلى الكائن EContact إذا كان موجوداً فى الذاكــرة من قبل، أو يمرر الطلب إلى الكائن الوســيط MBroker الذى يقوم بتفويض الكائن FReader لاسترجاع الكائن المطلوب من قاعدة البيانات.

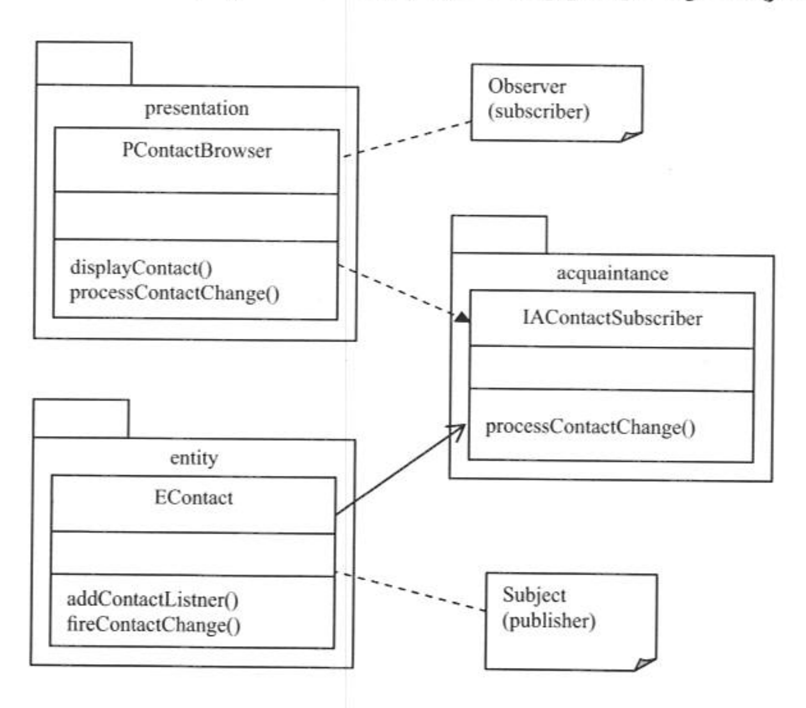


شكل (٦-١٥) نمط سلسلة المسؤوليات

٦-٣-٤ نمط المراقب Observer:

يعتبر الغرض من اســتخدام نمط المراقب هو تأسيس علاقة تبعية واحد - إلى - متعدد بيــن الكائنات، بحيث إذا حدث تغييــر فى الكائن فإنه يتم تحديث الكائنــات التابعة له آلياً. يســمى الكائن الأول كائن «Subject» أو «Publisher» وتســمى الكائنــات التابعة له كائنات

«Observers» أو «Subscribers». يتم إعلام الكائنات التابعة بأية تغييرات تحدث فى الكائن الأول وتقـوم بإجـراء التعديلات الضرورية لتلائم حالتها مع حالـة الكائن الأول. يؤدى هذا الأسـلوب إلى تقليـل درجة الاقتران بين الكائنـات الذى يحدث نتيجـة وجودها داخل حزم مختلفة فى شكل طبقات داخل إطار معمارى مثل PCMEF. تتم عملية الإعلام عن التغييرات فى شـكل سلسـلة من أسفل إلى أعلى داخل الإطار. يوضح شـكل (٦-١٦) مثالاً لاستخدام نمط المراقب، إذ يقوم الكائن EContact بإعلام الكائن PContactBrowser بالتغييرات التى حدثت له وذلك من خلال كائن وسيط نظراً لوجود الكائنات فى طبقات مختلفة.



شكل (٦-١٦) نمط المراقب

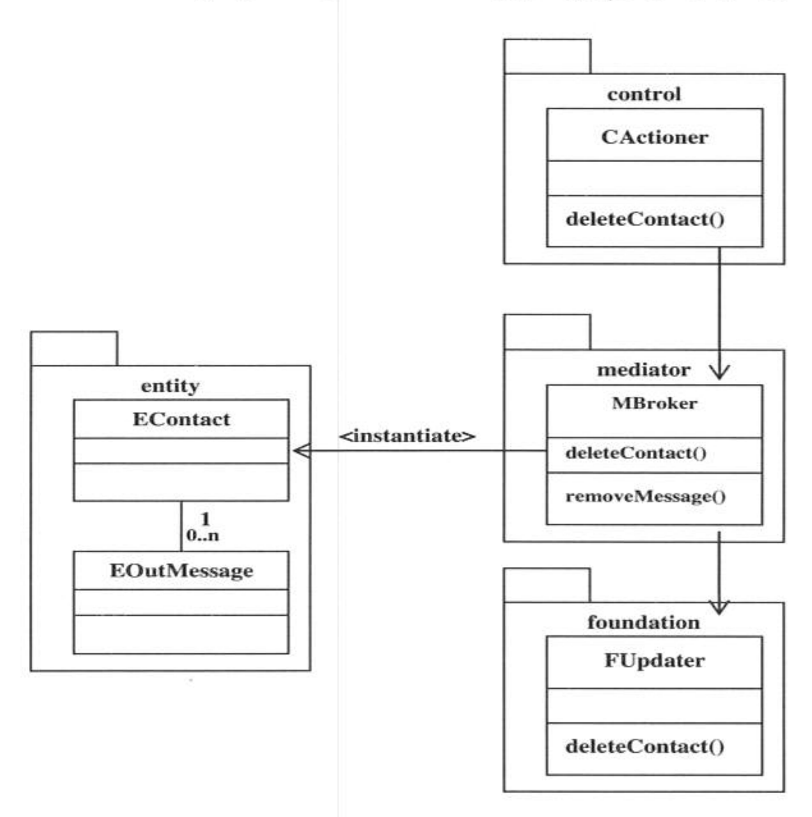
٦-٣-٥ نمط الوسيط Mediator:

يعتبر الغرض من اسـتخدام نمط الوسيط هو حل مشـكلة الاقتران المباشر بين الكائنـات الموجودة فى طبقات مختلفـة. يقدم هذا النمط مجموعـة من الكائنات الوسـيطة التى تقوم بعملية الاتصال بين الكائنـات الموجودة فى الطبقات المختلفة. لقـد تم اسـتخدام هذا النمط فى إطـار PCMEF، إذ تقوم طبقـة Mediator بدور

الوســيط بين طبقــة الأســاس Foundation وحزمة Entity. وتقــوم طبقة التحكم Control أيضاً بدور الوسيط بين طبقة العرض Presentation وحزمة Entity. يقوم نمط الوســيط بدور آخــر مهم وهو ضمان الحفاظ على التزامــن بين كائن Subject وكائنات Observers عند حدوث تغييرات فيه. يوضح شكل (٦-١٧) مثالاً لاستخدام نمط الوســيط، إذ يطلــب الكائن CActioner من الكائن الوســيط MBroker القيام بحذف أحد الكائنات EContact. يقوم الكائن الوسيط بمهمتين هما:

١- يمرر رســالة حذف الكائــن EContact إلى الكائــن FUpdate الموجود فى طبقة الأساس لتنفيذ عملية حذف الكائن من قاعدة البيانات.

٢- يقــوم بالتحقق من وجود الكائــن EContact فى الذاكرة، فــإذا كان موجوداً فإن الكائن الوسيط يقوم بإرسال رسالة لحذفه من الذاكرة أيضاً.



شكل (٦-١٧) نمط الوسيط



الفصل السابع
تصميم قواعد البيانات
Database Design

تعتبر البيانات Data هى الرافد الأساسى لنظم المعلومات، إذ يتم تخزينها فيما يسمى بقواعد البيانات Database ليتم التعامل معها من خلال برامج التطبيق. تقوم برامج التطبيق بالاتصال بقاعدة البيانات لاسترجاع البيانات المطلوب معالجتها وإجراء التعديلات عليها، ثم إعادة تخزينها مرة أخرى فى قاعدة البيانات. وأحياناً تقوم بعض برامج التطبيق بالاستعلام فقط عن بعض البيانات المخزنة بقواعد البيانات من دون إجراء تعديلات عليها. وعلى هذا فإنه يمكن للعديد من البرامج المختلفة أن تتعامل مع قاعدة بيانات معينة فى الوقت نفسه. أى أن قاعدة البيانات تقوم بدور الخادم Server للعديد من التطبيقات التى تقدم خدمات لعديد من المستخدمين من خلال محطات عمل مختلفة Client Workstations مرتبطة بقاعدة البيانات. هناك مجموعة من البرمجيات التى تمكننا من التعامل مع قواعد البيانات وتكون هى المسؤولة عن عمليات التخزين واسترجاع ومعالجة البيانات ويطلق عليها اسم «نظام إدارة قواعد البيانات (Database Management System (DBMS". وهناك أيضاً لغة برمجة قياسية تستخدم للتعامل مع قواعد البيانات يطلق عليها اسم «لغة الاستفسار الهيكلية (Structured Query Language (SQL".

تتسم قواعد البيانات الجيدة بالخصائص التالية:

- كبيرة الحجم Large Size.
- دائمة Persistent.
- قابلة للاشتراك للعديد من المستخدمين Multi-User Shareable.
- قابلة للاسترداد Recoverable.
- متسقة Consistent.
- آمنة Secure.
- قابلة للاتساع Extensible.

٧-١ قواعد البيانات المرتبطة Relational Databases:

تستخدم قواعد البيانات المرتبطة على نطاق واسع منذ أكثر من عشرين عاماً لتخزين ومعالجة بيانات مشاريع نظم المعلومات. يقوم نموذج البيانات المرتبط Relational Data Model بتمثيل البيانات فى شكل سجلات Records أو صفوف Rows داخل مجموعة من الجداول المرتبطة معاً فى شكل علاقات منطقية تربط بين الجداول عن طريق بعض الخصائص المشتركة. يعتمد هذا النموذج بشكل أساسى على مجموعة من العلاقات الرياضية بين المجموعات وكذلك العلاقات المنطقية بينها. سوف نستعرض الآن أهم خصائص هذا النموذج وكيفية تطبيقه وبرمجته مع الاستعانة بحالة دراسية لتوضيح المفاهيم.

٧-١-١ الجدول Table:

يعتبر الجدول هو أحد المكونات الأساسية فى نموذج قواعد البيانات المرتبطة. يتكون الجدول من مجموعة محددة من الأعمدة Columns وعدد غير محدد من الصفوف Rows. تمثل الأعمدة مجموعة الحقول أو الخصائص Attributes التى تميز عناصر الجدول. يوضح شكل (٧-١) هيكل أحد الجداول وهو يمثل بيانات الأفلام وذلك باستخدام نظام قواعد البيانات أوراكل ORACLE Database. يركز التصميم على تحديد مواصفات كل عمود فى الجدول مثل: الاسم Column Name ونوع البيانات للعمود Data Type ونوع المفتاح المستخدم PK، FK وإمكانية أن يكون العمود خالياً من البيانات Null.

movie			
movie_code	Number(5)	<pk>	not null
movie_title	Varchar2(30)		null
director	Varchar2(20)		null

شكل (٧-١) تصميم جدول الأفلام

وبتطبيق قواعد المجموعات الرياضية فإن الجدول يحتوى على مجموعة مميزة من الصفوف أى أنه لا يمكن تكرار بيانات أحد الصفوف داخل الجدول. ولضمان ذلك يتم استخدام أسلوب المفتاح الأساسى Primary Key (PK) للجدول. يتكون المفتاح

الأساســـى للجدول من عمود واحد أو أكثر وهى ذات قيم فريدة لا يمكن تكرارها أبداً داخل الجدول. ومن الطبيعى أن تكون هذه الأعمدة المكونة للمفتاح الأساسى غير خالية من البيانات NOT NULL. يمكننا تطبيق مواصفات التصميم للجدول باستخدام لغة الاستفســـار الهيكلية SQL أو باســـتخدام إحدى أدوات نظم إدارة قواعد البيانات مثل ORACLE Schema Builder. يوضح شـــكل (٧-٢) جملة SQL المســـتخدمة لإنشاء الجدول حسب التصميم المبين بشكل (٧-١).

Create Table movie

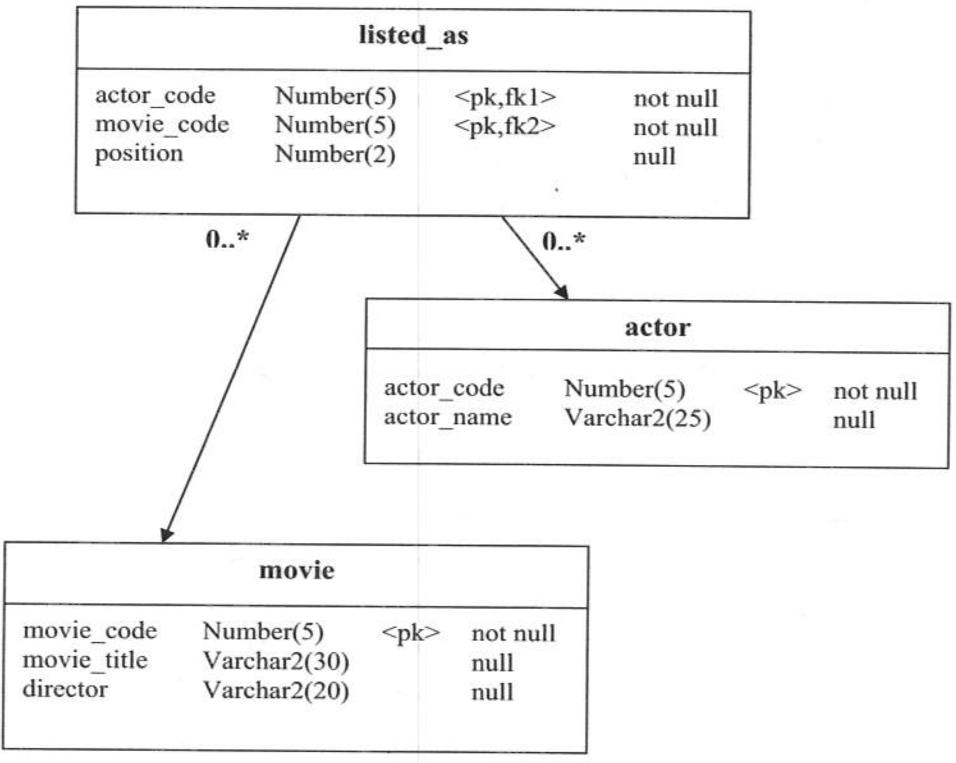
```
Create Table Movie (
        Movie_Code   Number(5)      Not Null ,
        Movie_Title   Varchar2(30) ,
        Director         Varchar2(20) ,
Constraint  PK_Movie  Primary Key (Movie_Code)
        ) ;
```

شكل (٧-٢) جملة SQL لإنشاء جدول الأفلام

٧-١-٢ التكامل المرجعى Referential Integrity:

تتميز قواعـــد البيانات المرتبطـــة Relational Databases بتطبيقها لمفهوم الربط بين الجداول باســـتخدام أســـلوب التكامل المرجعى. فمن المعـــروف أن عملية الربط بين الجداول تتم باســـتخدام نوعين مـــن المفاتيح هما المفتاح الأساســـى PK والمفتاح الخارجى FK، إذ يتم ربط أحد ســـجلات الجدول الذى يحتوى على المفتاح الأساســـى بجميع الســـجلات المقابلة له فى الجدول المرتبط به، وتكون قيمة المفتاح الخارجى له تساوى قيمة المفتاح الأساســـى. تكمن أهمية تطبيق مبدأ التكامل المرجعى إلى كونها تضمن سلامة بيانات الجداول المرتبطة، إذ إنه لا يمكن تسجيل بيانات فى الجدول إلا إذا كان يحتوى على قيمة للمفتاح الخارجى يوجد لها نظير مطابق للمفتاح الأساســـى فى الجدول المرتبط به. يوضح شـــكل (٧-٣) نموذجاً منطقيـــاً لقاعدة بيانات تتكون مـــن ثلاثة جداول مرتبطة. يحتوى جـــدول Movie على بيانات الأفلام فقط ويحتوى جـــدول Actor على بيانات الممثلين، فى حين يقـــوم جدول Listed-As بدور مهم جداً

وهو تجميع البيانات المرتبطة بكلا الجدولين، حيث يحتوى على بيانات دور كل ممثل فى الأفلام التى اشترك فيها. نلاحظ فى المثال أن المفتاح الأساسى للجدول Listed-As يتكون من المفتاحين الخارجيين Fk1,Fk2 فى شكل مفتاح أساسى مركب، إذ إنه لا يمكن تسجيل بيانات تمثل اشتراك الممثل نفسه فى الفيلم نفسه مرة أخرى. يتضح من العلاقة أن الفيلم الواحد يحتوى على مجموعة من الممثلين، كما أن كل ممثل يمكن أن يشارك فى أكثر من فيلم. ونلاحظ أيضاً أن قيم كل من المفاتيح الأساسية والمفاتيح الخارجية لا يمكن أن تكون خالية Not Null.

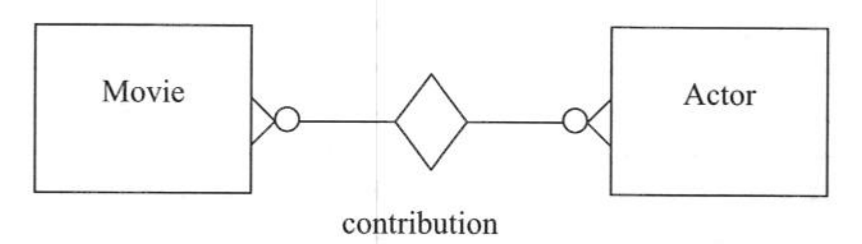


شكل (٧-٣) نموذج منطقى لقاعدة بيانات الأفلام

٧-١-٣ النموذج المفاهيمى Conceptual Model:

تمر عملية نمذجة قواعد البيانات بعدة مستويات من التجريد Abstraction. يعتمد كل مستوى على درجة التفاصيل التى يجب أن يقدمها النموذج. تتسم نمذجة

بيانـــات النظام فى المراحــل الأولية للتصميم باعتمادها الأســلوب المفاهيمى لبيانات كائنات النظام والعلاقات الطبيعية بينها دون الدخول فى التفاصيل الخاصة بســلامة البيانـــات وتكاملهـــا أو حتى إمكانية تطبيقها باســتخدام إحــدى أدوات إدارة قواعد البيانات. يعتبر مخطط العلاقة بين الكائنات (Entity Relationship Diagram (ERD هو أحد أساليب نمذجة قواعد البيانات بشكل مفاهيمى. يوضح شكل (٧-٤) النموذج المفاهيمــى المقابــل للنموذج المنطقى الموضح بشــكل (٧-٣). نلاحظ من الشــكل أن النمــوذج قد أخفى بعض التفاصيل التى تخــص عملية التكامل المرجعى بين الجداول وأنه يركز فقط على إظهار البيانات الأساســية لكل جــدول وتوضيح أنواع العلاقات الطبيعية بين الجداول.



شكل (٧-٤) النموذج المفاهيمى لقاعدة بيانات الأفلام

٧-١-٤ النموذج المنطقى Logical Model:

يهتم النموذج المنطقى بتقديم نموذج حقيقى قابل للتطبيق باستخدام إحدى أدوات إدارة قواعد البيانات. وعلى هذا فهو يهتم بما يلى:

- كائنات الربط Association Entity:

هــى عبارة عن كائنات تمثل العلاقة بين كائنات أخرى وخصوصاً إذا كانت العلاقة تحتوى على خصائص تميزها. تستخدم هذه الكائنات أيضاً لتبسيط بعض العلاقات المركبــة التى يصعب تطبيقها فعلياً باســتخدام الأدوات المتاحة لإدارة قواعد البيانات مثل علاقة متعدد - إلى - متعدد أو العلاقة بين ثلاثة كائنات أو أكثر.

- التكامل المرجعى Referential Integrity:

هو كما ذكرنا سابقاً يهتم بسلامة البيانات الموجودة فى الجداول المرتبطة.

- نوعية نظام إدارة قواعد البيانات DBMS:

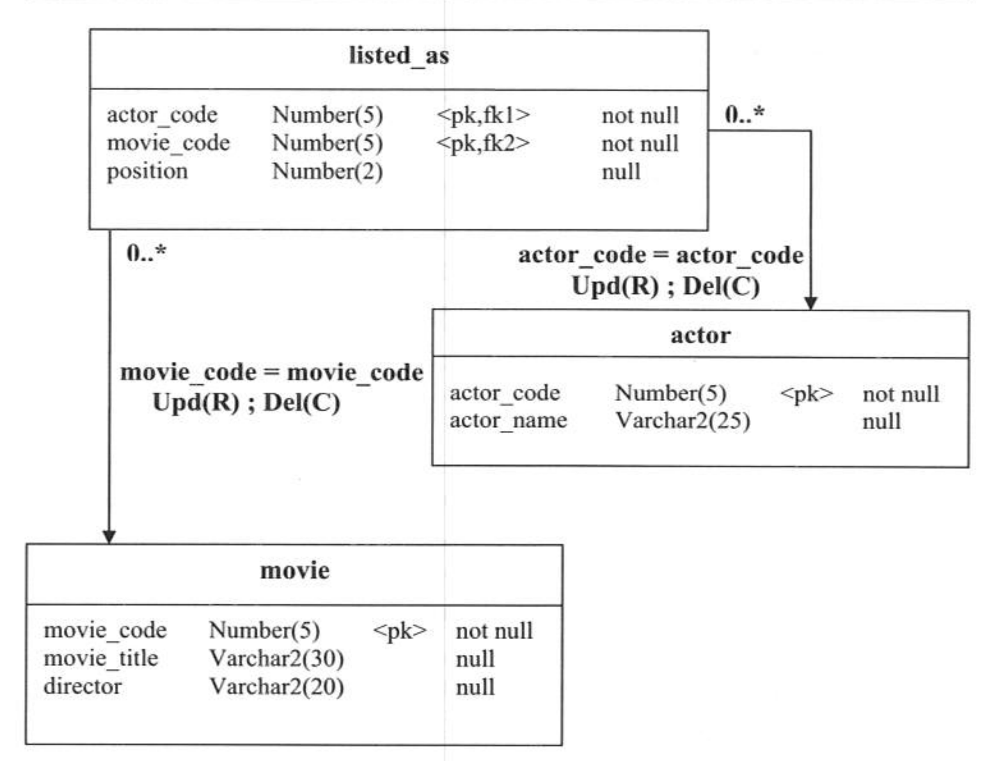
يتم تمثيل الجداول بالشكل الذى يلائم نوعية نظام إدارة قواعد البيانات المستخدم. يمثــل النموذج الموضح بالشــكل (٧-٣) أحد النماذج المنطقيــة لقاعدة بيانات الأفلام باستخدام نظام إدارة قواعد البيانات «أوراكل».

٧-١-٥ تطبيق قواعد العمل Implementing Business Rules:

مــع أنه قد تم الأخذ فى الاعتبار عند تصميم قواعد البيانات التأكد من ســـلامة بيانـــات الكائنات المرتبطة من خلال قيود التكامل المرجعى، إلا أنه يجب أيضاً التحقق مـــن صحة البيانات من وجهة نظر نظام الأعمال وذلك من خلال تطبيق بعض قواعد العمــل الضرورية الخاصـــة بطبيعة نظام الأعمال. أحياناً تكــون بعض قواعد العمل بســيطة ويمكن تطبيقها على جميع الســجلات بالجدول مثل إســناد قيمة افتراضية لأحد الحقول عند إنشـــاء ســجل جديد (كأن يتم إســناد قيمة تاريخ اليوم إلى حقل التاريــخ بالجدول). وكذلك إضافــة قيد على عملية الحذف لأحد ســجلات جدول مرتبــط بآخر. فمثلاً يمكننا إضافة قيد عمل خاص بقاعدة بيانات الأفلام يؤدى إلى حذف جميع الســجلات الموجودة فى جــدول Listed_As والخاصة بفيلم معين قد تم حذفه من جدول Movie.

يتم تمثيل معظم قواعد العمل البســيطة من خلال تعريف قواعد التكامل المرجعى فى أثناء عمليات الإضافة والتعديل والحذف. يوضح شــكل (٧-٥) كيفية تمثيل بعض قواعد العمل البســيطة مثل Del (C) التى تعنى أنه فى حالة حذف ســجل من جدول الأفلام يجب امتداد الحذف ليشمل السجلات المتعلقة به فى الجدول المرتبط به. أما Upd (R) فتعنى أن عملية التعديل تكون مقيدة، إذ لا يسمح نظام إدارة قواعد البيانات بتعديل كود فيلم معين فى جدول الأفلام وذلك للحفاظ على ســـلامة التكامل المرجعى بينها وبين الجداول المرتبطة به.

هنـــاك أيضاً بعض قواعد العمل المركبـــة التى تتضمن تنفيذ بعض الإجراءات على البيانـــات المرتبطة بحدث معـــين. يتم تمثيل قواعد العمل المركبة فى شـــكل وحدات برمجية باســـتخدام إحدى لغات برمجة قواعد البيانات تسمى الزنادات «Triggers». يتـــم تطبيق هذه الوحـــدات البرمجية على الجدول المخصص لـــه عند حدوث أى من العمليات الأساسية على قواعد البيانات مثل الإضافة والتعديل والحذف. يوضح شكل (٧-٦) مثالاً لإحدى الوحدات البرمجية (الزناد) التى يتم تطبيقها على جدول الأفلام عند حذف فيلم معين.



شكل (٧-٥) قواعد العمل فى النموذج المنطقى

Procedural referential integrity with delete trigger in MovieActor

```
Create trigger t_movie after delete
On movie for each row
Begin
        -- Delete all children in "Listed_As"
        Delete  listed_as
        Where  movie_code = :old.movie_code;
End;
```

شكل (٧-٦) زناد الحذف على جدول الأفلام

٧-١-٦ برمجة تطبيقات قواعد البيانات Programming Database Applications:

لا ينبغى النظر لقواعد البيانات باعتبارها مجموعة من الجداول الصماء ولكن فى حقيقة الأمر أنها تكون نشطة ومتفاعلة مع التطبيقات المرتبطة بها من خلال مجموعة

مجموعــة من البرامــج. بعض هذه البرامج يكون مخزناً فى قواعد البيانات نفســها والبعض الآخر يكــون مخزناً فى التطبيقات التى تســتخدم قواعد البيانات. تتطلب برمجة تطبيقات قواعد البيانات استخدام إحدى لغات البرمجة التى تتضمن تعليمات لغة الاستفســار البنائية SQL. يتم إرســال كل جملة SQL إلى قواعد البيانات ليتم التحقق من صحتها وتفســيرها ومن ثم تنفيذها واســترجاع النتائج إلى التطبيق. إذا كان هناك العديد من جمل SQL التى تتطلب التعامل مع قواعد البيانات بشكل مستمر فسوف يؤثر ذلك فى أداء النظام بشكل سلبى. وعلى ذلك فإنه يجب عدم التوسع فى استخدام هذا الأسلوب إلا فى الحالات التى تتطلب أن تتم برمجة قواعد البيانات من خلال واجهة التطبيق. يوضح شكل (٧-٧) مثالاً لهذا الأسلوب.

استخدام تعليمات SQL للبحث فى قواعد البيانات من داخل تطبيق الأفلام

```
-- استعراض اسم الممثل صاحب دور البطولة لكل فيلم
Execute Select  movie_title , actor_name
          From  movie  m , listed_as  l , actor  a
           Where  m.movie_code = l.movie_code
            AND  l.actor_code = a.actor_code
            AND  l.position = 1 ;
```

شكل (٧-٧) برمجة قواعد البيانات من خلال التطبيق

يتــم فى أغلب الأحــوال برمجة تطبيقات قواعد البيانــات من خلال مجموعة من الإجــراءات المخزنــة Stored Procedures فى خــادم قواعد البيانات نفســها ويتم اســتدعاؤها من خلال واجهات التطبيق. يوضح شــكل (٧-٨) مثالاً لأحد الإجراءات المخزنة باســتخدام لغة أوراكل وذلك للبحث فى قاعــدة بيانات الأفلام. نلاحظ من الشكل أن تنفيذ الإجراء يتم من خلال بيئة برنامج SQL ولكن فى الحقيقة يكون جزءاً من برامج التطبيق المخزن عند العميل. ونلاحظ أيضاً أن الإجراء يحتوى على معامل إدخال Input Parameter يستخدم لتمرير متغيرات من التطبيق إلى الإجراء فى أثناء التنفيذ. ويتم أيضاً استرجاع نتائج تنفيذ الإجراء إلى التطبيق المستدعى له.

استخدام أسلوب الإجراءات المخزنة للبحث فى قواعد بيانات الأفلام

```
--

 Create or Replace Procedure string_search(string IN varchar2)  AS
 Cursor c1 IS
              Select movie_title AS found
              From movie
              Where upper(movie_title) Like '%' || upper(string) || '%';
       Begin
              For c1rec IN c1 Loop
                     Dbms_output.put_line
                     ('Found movie title: ' || LPAD(clrec.found,30));
              End Loop;
       End string_search;

SQL> Execute string_search( 'الخوف' );
Found  movie title :   شىء من الخوف
```

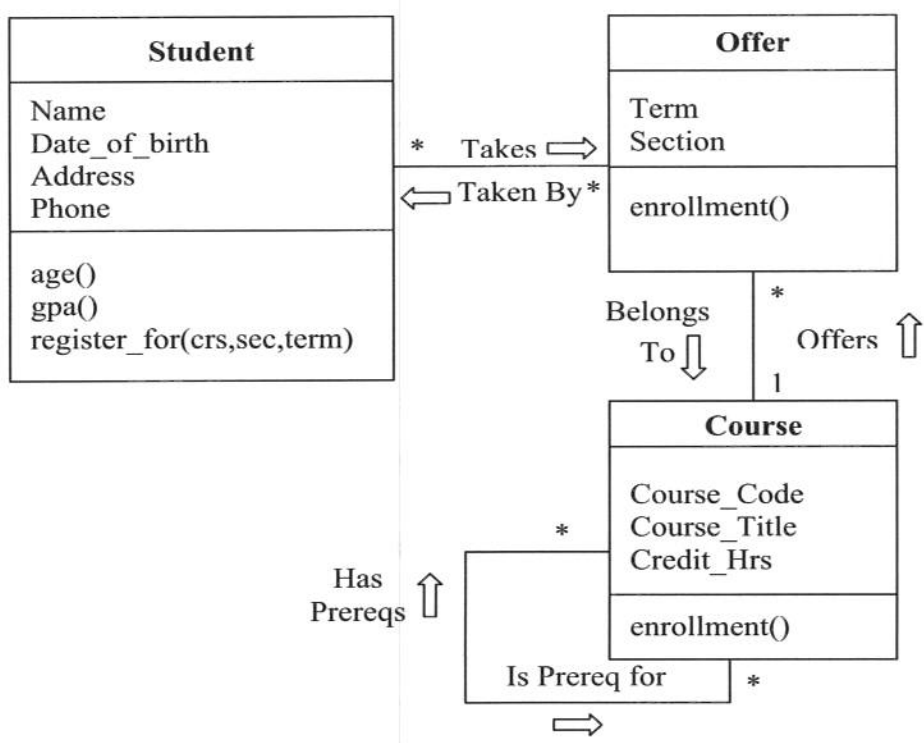
شكل (٧-٨) استخدام الإجراءات المخزنة Stored Procedures

٧-٢ قواعد البيانات الشيئية Object Databases:

مع زيادة استخدام أسلوب النمذجة الشيئية المعتمدة على الكائنات أصبح من الضرورى أن يتم اعتماد أسلوب لبناء قواعد بيانات معتمدة على الكائنات. وقد قامت مجموعة ODMG فى منتصف التسعينيات باعتماد معايير قياسية لقواعد البيانات المعتمدة على الكائنات OODB. وقد استحدثت المجموعة فى الفترة الأخيرة معايير أخرى تقوم على التحول من توصيف نظام إدارة قواعد البيانات المعتمدة على الكائنات إلى تعريف طبقة برمجية جديدة تسمى «Object Storage API» بين برامج التطبيق وقواعد البيانات المتصلة بها. تقوم هذه الطبقة بدور التناظر بين كائنات التطبيق وبين السجلات فى جداول قواعد البيانات. يتم اعتماد هذا الأسلوب ليلائم الإجراءات الروتينية المستخدمة فى إدارة قواعد البيانات.

تقدم مجموعة ODMG لغة استفسار معتمدة على الكائنات تسمى (Object Query Language (OQL مشابهة للغة الاستفسار البنائية SQL المستخدمة فى قواعد البيانات المرتبطة. يمكننا أيضاً اســتخدام لغة تعريف الكائنات (Object Definition Language (ODL لتوصيف هيكل قواعد البيانات المعتمدة على الكائنات (Hoffer et al, 2002).

ولتوضيح كيفية اســتخدام هذه اللغة سوف نســتعين بمثال لنمذجة الأصناف فى تطبيق التحاق الطلاب بالجامعة باستخدام لغة النمذجة الموحدة UML كما هو موضح بالشكل (٧-٩). يمثل هذا المخطط النموذج المفاهيمى لقواعد بيانات النظام.



شكل (٧-٩) النموذج المفاهيمى لقواعد بيانات الجامعة

تستخدم لغة ODL بعض الكلمات المحجوزة لتعريف كل من خصائص Attributes الكائنات والعلاقات بينها. وتســتخدم أيضاً الشــكل العام للدوال فى تعريف الطرق Methods. يوضح شــكل (٧-١٠) اســتخدام لغة ODL فى توصيــف قواعد بيانات الجامعة المقابل للنموذج المفاهيمى الموضح فى شكل (٧-٩).

```
Class Student {
      ( extent students )
        Attribute string  name;
        Attribute date date_of_birth;
        Attribute  Address  address;
        Attribute  Phone  phone;
        Relationship  set (Offer) takes inverse Offer::taken_by;
        Short  age();
        Float  gpa();
        Boolean  register_for(string  crs, short  sec, string  term);
};

Class Offer  {
     ( extent  offers)
        Attribute string  term;
        Attribute enum  section {1,2,3,4,5,6,7,8};
        Relationship  set (Student) taken_by inverse Student::takes;
        Relationship  Course  belongs_to  inverse  Course::offers
        Short  enrollment();
};

Class  Course {
      ( extent  courses)
        Attribute  string course_code;
        Attribute  string course_title;
        Attribute  short credit_hrs;
        Relationship set (Course) has_prereqs  inverse Course::is_prereq_for;
        Relationship set (Course) is_prereq_for inverse Course::has_prereqs;
        Relationship list (Offer) offers inverse Offer:belongs_to;
        Short  enrollment();
};
```

شكل (٧-١٠) استخدام لغة ODL لتوصيف قاعدة بيانات الجامعة

٣-٧ قواعد البيانات الشيئية-المرتبطة Object-Relational Databases:

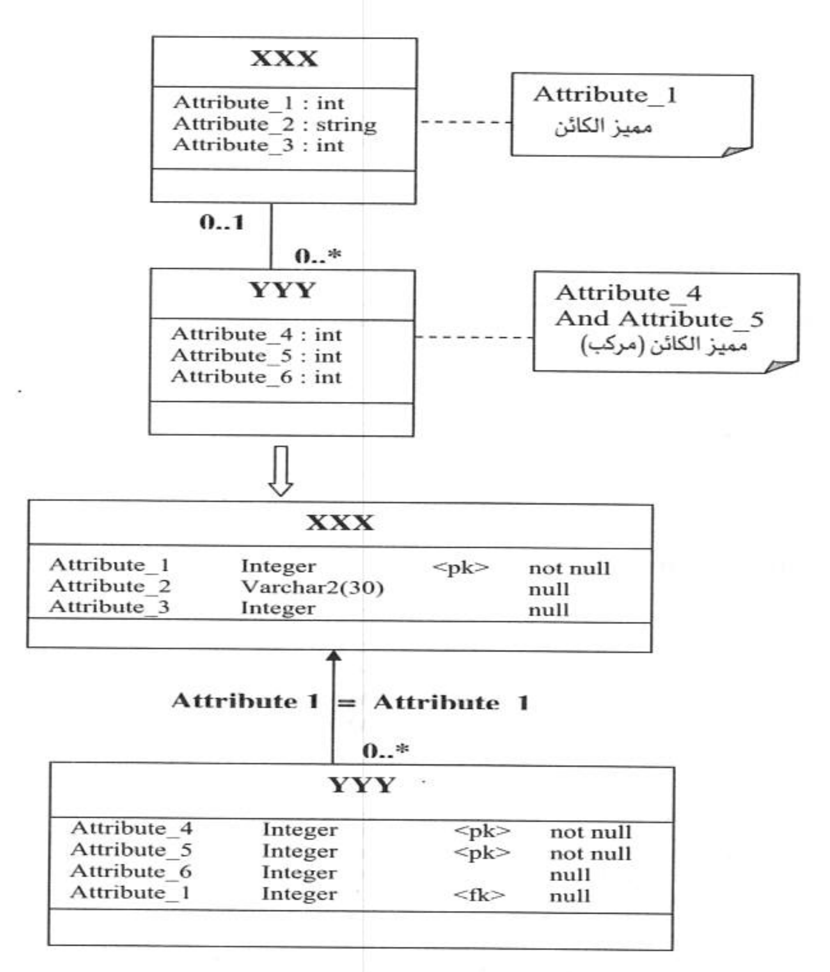
تحتوى كل من قواعد البيانات المرتبطة RDB وقواعد البيانات الشيئية ODB على مجموعـــة من المميزات ونقاط الضعف. ويلاحظ أن نقاط الضعف فى أى منها تمثل

مميزات النوع الآخر. وبالطبع فقد أدى ذلك إلى ظهور نوع جديد من قواعد البيانات يسمى (ORDB) Object-Relational وهى تحتوى على مميزات كل من النوعين السابقين. تعتبر النسخة ORACLE 9I لقواعد البيانات أوراكل هى أحد أنواع نظم إدارة قواعد البيانات الشيئية-المرتبطة ORDBMS. يدعم هذا النوع من قواعد البيانات عمليات تعريف كل من البيانات المرتبطة Relational Data والكائنات Objects، بالإضافة إلى إمكانية التعامل معها والاستفسار عنها وذلك من خلال لغة موحدة وواجهة مشتركة. يأخذ النموذج الأساسى للبيانات الشكل العلاقى (الارتباطى Relational)، فى حين تأخذ العمليات والإجراءات الشكل الشيئى (المبنى على الكائنات Object-Oriented). تتم عمليات تخزين واسترجاع البيانات باستخدام قاعدة بيانات مرتبطة. ويتطلب ذلك إجراء عملية تحويل تناظرى بين الأسلوبين (Object, Relational). وقد استلزم ذلك إصدار نسخة مطورة من لغة الاستفسار لتلائم طبيعة التعامل مع كل من نوعى البيانات. يدعم هذا الأسلوب استخدام المفاهيم الأساسية للأسلوب الشيئى مثل: التوريث والتجريد واستخدام أنواع معقدة من البيانات مثل ملفات الصوت والصورة وغيرها، بالإضافة إلى دعمه لأسلوب التكامل المرجعى للتأكد من سلامة البيانات المخزنة فى الجداول المرتبطة. وجدير بالذكر أن عملية التحويل بالتناظر بين الكائنات والجداول المرتبطة يمكن أن تتم فى الاتجاهين. فمن الممكن أن يتم التحويل من الكائنات إلى الجداول المرتبطة وذلك فى عند إنشاء قاعدة البيانات بناءً على نموذج الأصناف للتطبيق. ويمكن أن يتم التحويل من الجداول المرتبطة إلى الكائنات وذلك فى أثناء معالجة البيانات المستخرجة من قاعدة البيانات إلى ذاكرة التطبيق. غالباً ما تتم عملية تحويل الكائنات إلى الجداول المناظرة لها بسهولة أكبر من عملية تحويل العلاقات بين الكائنات إلى العلاقات المناظرة لها بين الجداول. سوف نستعرض الآن عملية تحويل العلاقات الأكثر استخداماً فى نمذجة قواعد البيانات.

تحويل علاقة واحد - إلى - متعدد:

تعتبر علاقة واحد - إلى - متعدد من أكثر العلاقات شيوعاً ومن أكثرها سهولة أيضاً فى أثناء عملية التحويل إلى العلاقة المقابلة بين الجداول. يتم تحويل الكائن ناحية طرف العلاقة الأحادية إلى جدول يحتوى على مجموعة من الأعمدة المقابلة لمجموعة خصائص الكائن Attributes مع تحديد المفتاح الأساسى الذى يميز جدول

الكائن. ويتم تحويل الكائن ناحية طرف العلاقة المتعددة إلى جدول آخر يضم مجموعة خصائص الكائن مـع إضافة عمود يمثل المفتاح الخارجي للجدول FK يرتبط بالمفتاح الأساسى فى الجدول السابق. يوضح شكل (٧-١١) مثالاً على هذا التحويل.

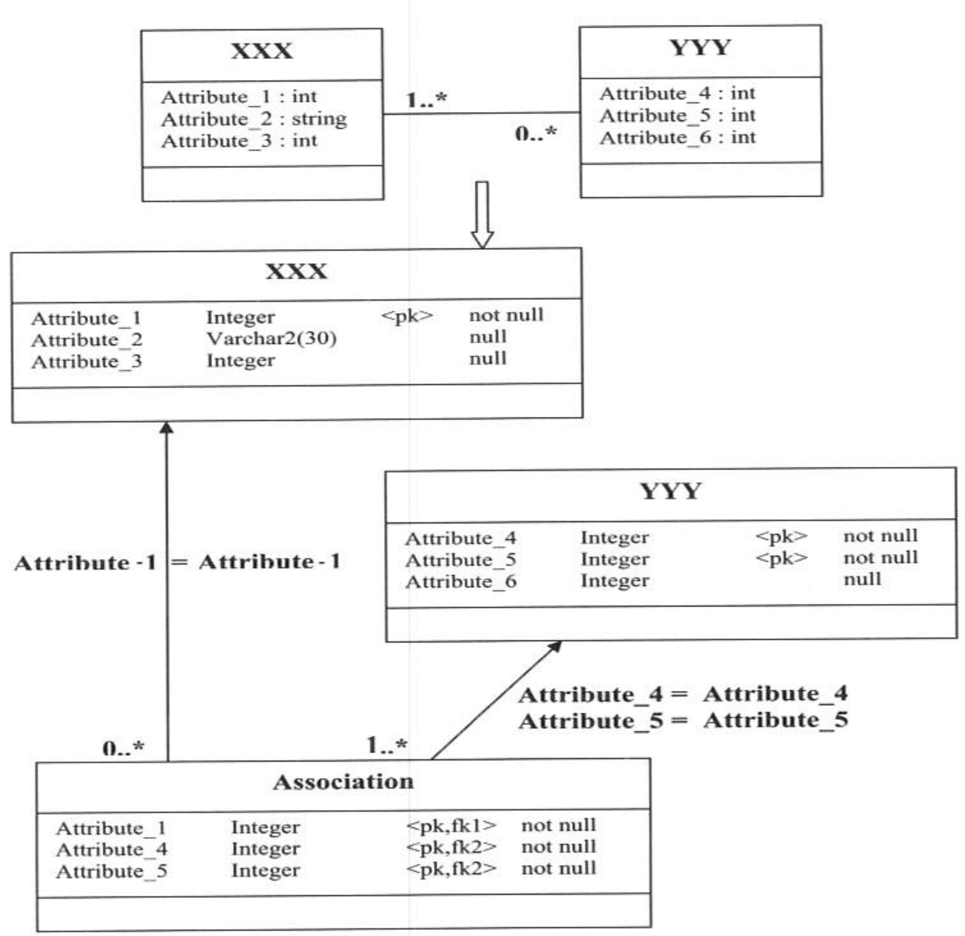


شكل (٧-١١) تحويل علاقة واحد - إلى - متعدد

تحويل علاقة متعدد – إلى – متعدد:

من المعروف أن قواعد البيانات المرتبطة لا تسـتطيع تطبيق العلاقة متعدد – إلى – متعـدد مباشـرة، ولكن يتم إضافة جدول خاص لتمثيل تلـك العلاقة وتحويلها إلى

علاقتين من النوع واحد - إلى - متعدد. يحتوى جدول العلاقة على مجموعة من الأعمدة تمثل المفاتيح الخارجية المقابلة للمفاتيح الأساسية فى الجدولين الأساسيين كما هو موضح بالشكل (٧-١٢).

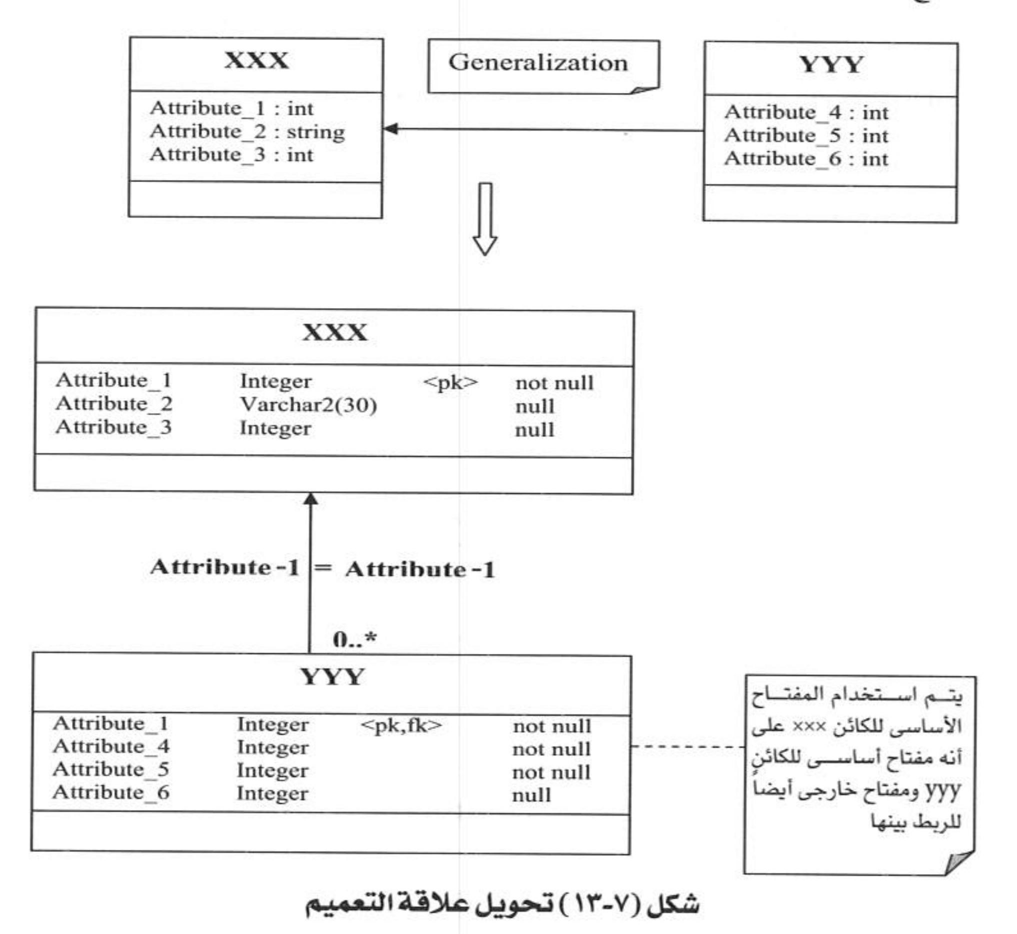


شكل (٧-١٢) تحويل العلاقة متعدد - إلى - متعدد

تحويل علاقة التعميم:

لا يستطيع النموذج العلاقى أن يدعم مفهوم التوريث الخاص بالنموذج الشيئى. ولحل هذه المشكلة يمكننا أن نكرر جميع الخصائص الموجودة فى الصنف العام

Superclass وتضمينهـا فـي الجدول الذى يمثل الصنف الفرعـي Subclass ونكتفى بهـذا الجدول الذى يمثل كلاً مـن الصنف العام والصنف الفرعـى. هناك حل آخر موضح بالشكل (٧-١٣) وهو يقوم على تحويل الصنف العام إلى جدول مماثل وتحويل الصنف الفرعى إلى جدول آخر مع إضافة عمود يمثل المفتاح الأساسـى للجدول وهو المفتاح الأساسى نفسه بالجدول الأول.

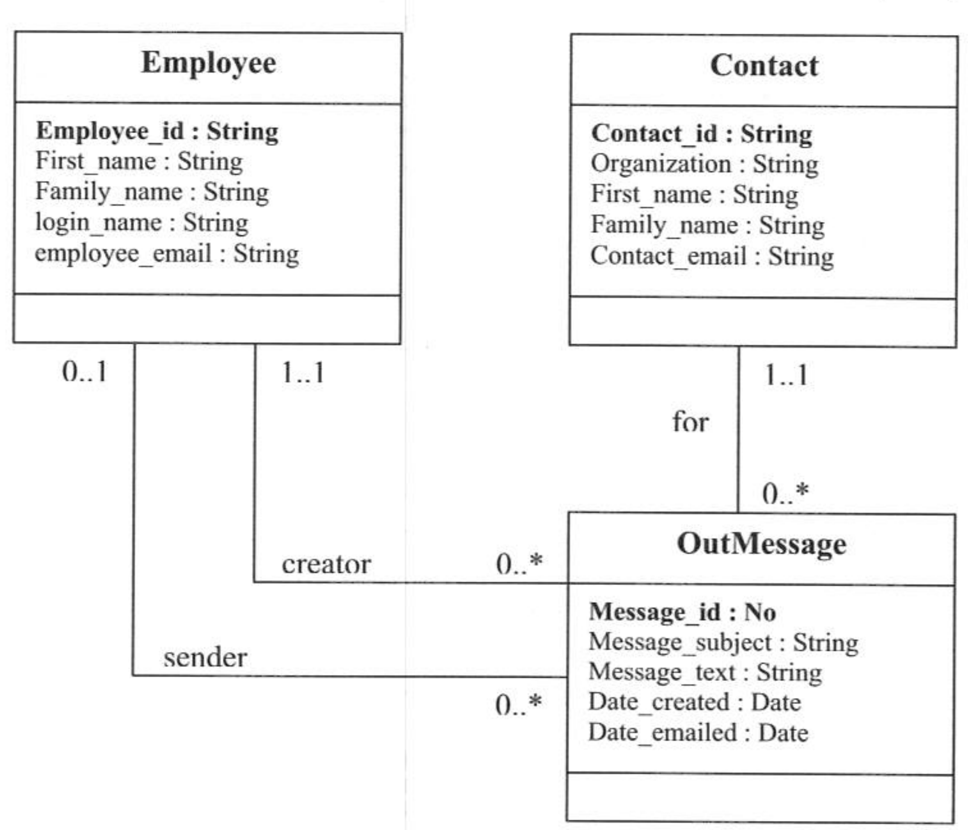


شكل (٧-١٣) تحويل علاقة التعميم

٧-٤ تصميم قواعد البيانات Database Design:

سنقوم الآن بتصميم قواعد البيانـات الخاصة بالحالة الدراسـية (إدارة البريد الإلكترونـى EM) التى بدأناها فـى الفصل الخامس من هذا الكتـاب. تقوم برامج

الحالة الدراســية بعمليات استخراج البيانات من قواعد البيانات وتجهيزها ثم إرسال الرســائل للعمــلاء، وبعد ذلك يتم تحديــث قواعد البيانات بتمييز الرســائل التى تم إرســالها فعلاً وذلك بإضافة معلومات عن عملية الإرســال مثل الوقت. يوضح شكل (٧-١٤) مخطط الأصناف للتطبيق EM.



شكل (٧-١٤) مخطط الأصناف للتطبيق EM

٧-٤-١ نموذج قواعد البيانات Database Model:

يتم اســتنتاج نموذج قواعد البيانات من نموذج الأصناف للتطبيق بطريقة مباشرة وسهلة كما هو مبين بشكل (٧-١٥). يمثل هذا المخطط النموذج المادى لقواعد بيانات التطبيق Physical Database Model، حيث إنه يحدد الخصائص ونوعها باســتخدام نظام معين لإدارة قواعد البيانات وهو نظام «أوراكل».

Employee

Employee_id	**char(4) <pk>**	**not null**
First_name	varchar2(20)	null
Family_name	varchar2(40)	not null
login_name	varchar2(40)	not null
employee_email	varchar2(60)	not null

Contact

Contact_id	**char(5)**	**<pk>**	**not null**
Organization	varchar2(80)		null
First_name	varchar2(20)		null
Family_name	varchar2(40)		not null
Contact_email	varchar2(60)		null

contact_id = contact_id

0..*

employee_id =
creator_emp_id

0..*

employee_id =
sender_emp_id

0..*

OutMessage

Message_id	**Integer**	**<pk>**	**not null**
Contact_id	Char(5)	<fk1>	not null
Creator_emp_id	Char(4)	<fk2>	not null
Sender_emp_id	Char(4)	<fk3>	null
Message_subject	Varchar2(40)		not null
Message_text	Varchar2(255)		not null
Date_created	Date		not null
Date_emailed	Date		null

شكل (٧-١٥) النموذج المادى لقواعد بيانات التطبيق EM

٧-٤-٢ إنشاء مخطط قواعد البيانات Creating Database Schema:

الخطـوة التاليـة لتصميم قواعد البيانـات تتضمن مجموعـة تعليمات لغة SQL الخاصة بإنشـاء الجداول والعلاقات بينها بنـاء على النموذج المادى لقواعد البيانات. يمكننا اسـتخدام إحدى أدوات هندسة البرمجيات Case Tools لتوليد الملف الخاص بتعليمات إنشاء قواعد البيانات من نموذج قواعد البيانات. يتم تنفيذ هذا الملف لإنشاء هيكل الجداول المرتبطة بقاعدة البيانات للتطبيق. يوضح شكل (٧-١٦) محتوى ملف تعليمات SQL الخاصة بإنشاء مخطط قواعد البيانات للتطبيق EM.

ملف يحتوى على تعليمات SQL لإنشاء قواعد بيانات التطبيق EM

```
-- Table : Contact
Create table Contact (
        Contact_id     char(5)              not null,
        Organization   varchar2(80),
        First_name     varchar2(20),
        Family_name    varchar2(40)         not null,
        Contact_email  varchar2(60),
        Constraint PK_CONTACT  primary key (contact_id) )
/
-- Table :  Employee
Create table Employee (
        Employee_id    char(4)              not null,
        First_name     varchar2(20),
        Family_name    varchar2(40)         not null,
        Login_name     varchar2(40)         not null,
        employee_email varchar2(60)         not null,
        Constraint PK_EMPLOYEE  primary key (employee_id) )
/
-- Index : Login_UN
Create unique index login_UN on Employee (
        Login_name  ASC  )
/
-- Table : Outmessage
Create table outmessage  (
        Message_id          integer         not null,
        Contact_id          char(5)         not null,
        Creator_emp_id      char(4)         not null,
        Sender_emp_id       char(4),
        Message_subject     varchar2(40)  not null,
        Message_text        varchar2(255)not null,
        Date_created        date            not null,
        Date_emailed        date,
Constraint PK_OUTMESSAGE primary key (message_id),
Constraint FK_OUTMES_REF_CONTACT foreign key (contact_id)
references  Contact (contact_id),
Constraint FK_OUTMES_REF_CREATOREMP foreign key
(creator_emp_id)  references  Employee (employee_id),
Constraint FK_OUTMES_REF_SENDEREMP foreign key (sender_emp_id)
references  Employee (employee_id)  )
/
```

شكل (٧-١٦) ملف تعليمات SQL لإنشاء قواعد بيانات التطبيق EM

٧-٤-٣ محتوى عينة قواعد البيانات Sample Database Content:

بعد إنجاز عملية إنشـــاء مخطـــط قواعد البيانات بنجاح، ســـنبدأ بتحميل قواعد البيانات بالبيانات الفعلية وذلك فى أثناء تنفيذ التطبيق. يمكننا أيضاً تحميل الجداول بالبيانات عن طريق تعليمات الإضافة الخاصة بلغة SQL. يوضح شكل (٧-١٧) بعض تعليمات SQL التى تقوم بالاستعلام واسترجاع البيانات الخاصة بالتطبيق EM.

```
SQL> select * from E,ployee;

Employee_id  First_name  Family_name  Login_name  Employee_email
-----------  ----------  -----------  ----------  --------------
1001         Ahmed       Al_Ghamdy    GhamdyA     Ghamdy@ipa.edu.sa
1002         Nasser      Al_Meshary   MesharyN    Meshary@ipa.edu.sa
1003         Alaa        Al_Sammak    SammakA     Sammak@ipa.edu.sa

SQL> select * from Contact;

Contact_id   Organization  First_name  Family_name  Contact_email
----------   ------------  ----------  -----------  -------------
1234         STC           Ahmed       Hesham       HeshamA@stc.net.sa
1235         KSU           Saleh       Al_Zahrany   Zahrany@ksu.edu.sa
1236         KFU           Kamal       Al_Dahshan   Dahshan@kfu.edu.sa

SQL> select contact_id, creator_emp_id, sender_emp_id, messaga_subject,
message_text  from  Outmessage  order by  contact_id;

Contact_id   Creator_emp_id   Sender_emp_id   Message_subject
----------   --------------   -------------   ---------------
Message_text
--------------------------------------------------------------
  1234      1001                              استعلام عن خدمات جديدة:
الرجــاء التكرم بموافاتنا بمعلومات مفصلة عن الخدمات الجديدة التى تقدمها الشــركة للعمــلاء، مع توضيح المزايا الخاصة بكل
خدمة وأسعار تقديمها.
  1235      1002                              طلب تقديم بحث:
مرفق مع الرسالة نسخة من البحث المقدم للنشر فى مجلة البحوث العلمية الخاصة بالجامعة، يرجى إحاطتنا بنتيجة فحص البحث
فور الانتهاء من تقييمه.
  1236      1003                              استعلام عن برنامج:
حيث إننى أنهيت دراســة البكالوريوس، تخصص علوم الحاســب، بدرجة جيد جداً، وأود الالتحاق ببرنامج الدراسات العليا لديكم
لدراسة الماجستير فى التخصص نفسه، أود التكرم بتزويدنا بتفاصيل البرنامج ومواعيد الالتحاق ورسوم الدراسة.
```

شكل (٧-١٧) محتوى عينة قاعدة البيانات EM



الفصل الثامن
تصميم الأصناف والتفاعلات
Classes and Interactions Design

تركز أنشطة تصميم الأصناف على عمليات التأكد من تحقيق الأصناف للمتطلبات الوظيفيـــة المحددة فى نموذج حالات الاســتخدام، بالإضافة إلــى ملاءمتها للتصميم المعمارى للنظام. تبدأ عمليات تصميم الأصناف بتحديد الأصناف والأماكن المناســبة لهـــا داخل الحـــزم Packages. وبعد ذلك يتم تحديد خصائــص وعمليات الأصناف التى تحقق المواصفات المطلوبة فى نموذج حالات الاســتخدام للتطبيق. وبعد ذلك يتم تأســيس العلاقات والتبعيات بين الأصناف. فى الواقع لا يمكن فصل عملية تصميم التفاعلات عن تصميم الأصناف، فبمجرد انتهاء عملية تحديد الأصناف يمكننا البدء فوراً فى تصميم التفاعلات التى ســوف تســاعدنا فى تحديــد مواصفات الأصناف وخصوصاً العمليات.

٨-١ إيجاد الأصناف من حالات الاستخدام Finding Classes from Use Cases:

تتطلـــب عملية تصميم الأصناف أن تحقق مواصفات المتطلبات المذكورة فى نموذج حالات الاستخدام. ولذلك فإنه من الطبيعى أن يتم استخراج تلك الأصناف من الوثائق المرفقة بنماذج حالات الاســتخدام التى تحتوى على المواصفات التفصيلية للمتطلبات الوظيفية لكل حالة استخدام. هناك أسلوب يعتمد على قائمة المتطلبات الموجودة فى حالات الاســتخدام ويستخدمها فى اســتخراج الأصناف وذلك فى شكل جدول يتكون من خمسة أعمدة كما يلى:

١- رقم المتطلب Requirement Number.

٢- تعريف المتطلب Requirement Definition.

٣- اسم الصنف والحزمة التابع لها Responsible Package and Class Name.

٤- اسم العملية للمتطلب Operation Name.

٥- اسم الحزمة والصنف المتعاون Collaboration Package and Class.

يوضح شـــكل (٨-١) جزءاً من الجدول المســتخدم لإيجــاد الأصناف من متطلبات حالات الاستخدام للتطبيق الخاص بإدارة البريد الإلكترونى EM.

رقم المتطلب	تعريف المتطلب	الحزمة / الصنف	اسم العملية	الحزمة/الصنف المتعاون
R1	يعـرض النظام رسـالة توضيحيـة للموظـف لإدخـال اسم المستخدم username وكلمة المرور password	Presentation PConsole Control CActioner	displayLogin getUserInput login	Presentation PConsole Control CActioner Foundation FConnection
R2	يقـوم النظـام بتنفيذ عملية اتصال الموظف بقاعدة بيانات النظام	Foundation FConnection Foundation FReader	getConnection readEmployee	Acquaintance IAconstants Foundation FReader
R3	يقـوم النظام بعرض قائمـة الاختيـارات المتاحة للموظف	Presentation PConsole	displayMenu	
R4	عندمـا يختار الموظف الاختيار الرابع بالقائمة يتم الخروج من النظام	Presentation PConsole Control CActioner	getUserInput exit	Control CActioner Foundation FConnection
R5	عندمـا يختار الموظف الاختيـار الأول يتـم عـرض رسـالة بعدد الرسائل غير المرسلة، وعرض معلومات عن كل رسالة، ثم عرض قائمة الاختيارات مرة أخرى	Presentation PConsole Mediator MBroker Foundation FReader	getUserInput viewMessages retrieveMessages readMessages readContact	Presentation PConsole Acquaintance IAconstants Mediator MBroker Foundation FReader

R6	عندما يختار الموظف الاختيـار الثانى، يقوم الموظف بإدخال رقم الرسـالة، ثـم يقـوم النظام بعرض محتوى الرسـالة، ثـم عرض قائمة الاختيارات مرة أخرى	Presentation PConsole	getUserInput displayMessageText	Presentation PConsole Mediator MBroker
R7	عندما يختـار الموظف الاختيـار الثالـث، يقوم الموظـف بإدخـال رقم الرسالة، ثم يقوم النظام بإرسال الرسالة وتعديل قاعـدة البيانات وعرض رسـالة تأكيد بإرسـال الرسالة، ثم عرض قائمة الاختيارات مرة أخرى	Presentation PConsole Control CActioner Foundation FWriter Presentation PConsole	getUserInput prepareMessage sendMessage updateMessage displayConfirmation	Presentation PConsole Control CActioner Acquaintance IAconstants Foundation FWriter
R8	فى حالة إدخال الموظف بيانـات غيـر صحيحة لاسم المستخدم أو كلمة المـرور، يقـوم النظام بعـرض رسـالة خطـأ، ويسمح للموظف بإعادة الإدخال مرة أخرى (عدد المرات المسموح بها =٣)	Ref. R1 Presentation PConsole Control CActioner	displayLoginError exit	Presentation PConsole Ref. R1 Control CActioner Foundation FConnection
R9	فى حالة إدخال الموظف رقم اختيار غير صحيح، يقـوم النظـام بتجاهل الاختيـار ويعيـد عرض قائمة الاختيـارات مرة أخـرى أو الخـروج بعد ٣ محاولات خطأ	Presentation PConsole Control CActioner Ref. R1	getUserInput displayMenu logout	Presentation PConsole Foundation FConnection

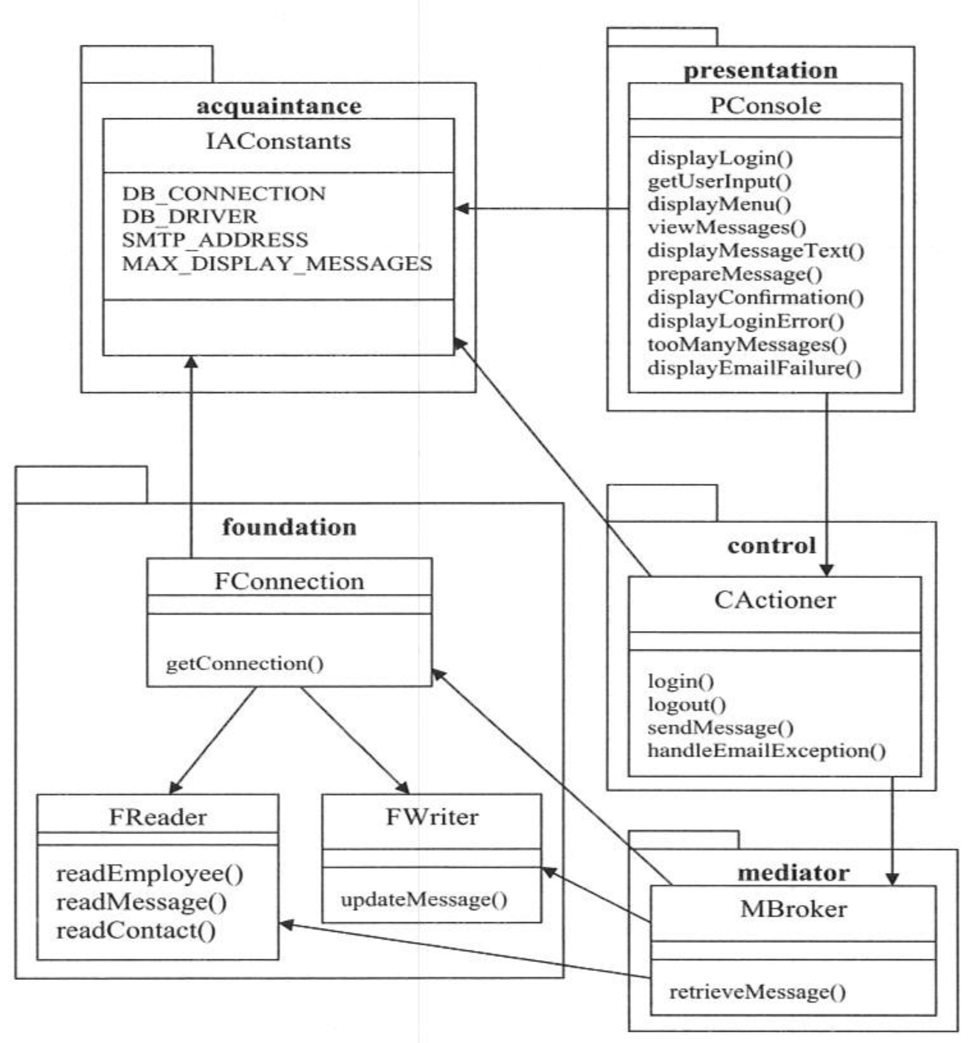
R10	فى حالة اختيار الموظف الاختيار الأول ويكون عدد الرسائل غير المرسلة أكبر من العدد المسموح بعرضها، يقوم النظام بعرض عدد الرسائل المسموح بها، ثم عرض رسالة توضيحية تفيد بوجود عدد آخر من الرسائل غير المرسلة موجودة بقاعدة البيانات	Ref. R5 Presentation PConsole	tooManyMessages	
R11	فى حالة فشل عملية إرسال الرسالة (الاختيار الثالث) يقوم النظام بعرض رسالة توضيحية للموظف تفيد بعدم نجاح عملية إرسال الرسالة، ثم يقوم بعرض قائمة الاختيارات مرة أخرى	Ref. R7 Control CActioner Presentation PConsole	andleEmailException displayEmailFailure displayMenu	Presentation PConsole

شكل (٨-١) جدول إيجاد الأصناف من حالات الاستخدام للتطبيق EM

٨-٢ تصميم الأصناف Classes Design:

يحتوى الجدول السابق على معلومات كافية لإعداد التصميم الأولى لنموذج الأصناف للتطبيق EM. يمثل النموذج الموضح بشكل (٨-٢) النموذج الأولى للأصناف والمقابل لجدول إيجاد الأصناف الموضح بشكل (٨-١). يعتبر هذا النموذج أولى لأنه لا يحتوى على جميع العلاقات بين الأصناف نتيجة عدم توافر جميع أنواع الرسائل المتبادلة بين الكائنات فى جدول إيجاد الأصناف. يتضمن النموذج الأولى الأصناف المذكورة فى جدول إيجاد الأصناف ومكانها داخل الحزم وكذلك علاقات الارتباط بينها. نلاحظ من النموذج أن بعض الحزم تحتوى على صنف واحد فقط لأن النموذج لا يتضمن الأصناف الخاصة بالكائنات الخارجية. يحتوى النموذج أيضاً على حزمة Acquaintance التى تحتوى على معظم الثوابت المستخدمة فى النظام مثل ثوابت الاتصال بقواعد البيانات Database__Connection التى يتم استخدامها من خلال صنف الاتصال FConnection والثوابت الخاصة بخادم البريد SMTP__Address التى يتم استخدامها من خلال صنف التحكم CActioner الخاص بالتطبيق، والثابت الخاص بأقصى عدد ممكن للرسائل التى يمكن عرضها على الشاشة Max__Display__Messages التى يتم استخدامها من خلال صنف الواجهة PConsole.

يحتاج هذا النموذج الأولى إلى بعض التحسينات والإضافات لتتوافق مع المتطلبات غيـر الوظيفية الموضحة فى إطار التصميم المعمارى للنظام. يؤدى ذلك إلى توسـعة النموذج الأولى استناداً إلى التصميم المعمارى وذلك بإضافة أصناف جديدة وتعديلات فى الأصناف المسـتندة إلى متطلبات المسـتخدم. تبدأ هذه العملية بجزء من جدول إيجاد الأصناف، وبعد ذلك يتم توسـعته ليضم أعمدة أخرى تقدم الأصناف الجديدة والتعديلات على الأصناف الموجودة، كما هو موضح بشكل (٨-٢).



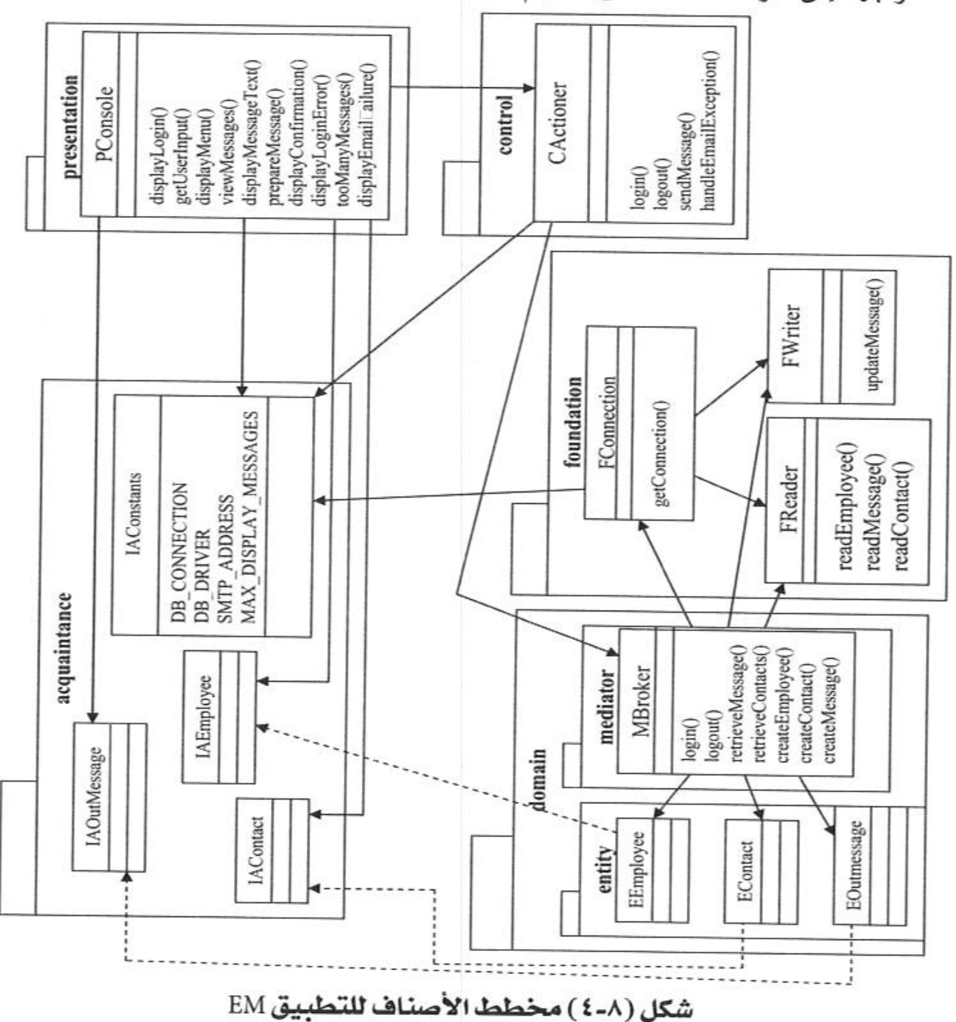
شكل (٨-٢) النموذج الأولى للأصناف للتطبيق EM

رقم المتطلب	الصنف/العملية	الحزمة/الصنف المتعاون	الإطار المعماري	الصنف/ العملية الجديد أو المعدل	الحزمة/الصنف المتعاون الجديد أو المعدل
R1	PConsole displayLogin PConsole getUserInput CActioner login	Presentation PConsole Control CActioner Mediator MBroker	DDP Chain of .Resp	MBroker login	Foundation FConnection
R2	FConnection connect FReader readEmployee	Acquaintance IAconstants Foundation FReader		MBroker createEmployee	entity EEmployee
R3	PConsole displayMenu				
R4	PConsole getUserInput CActioner exit	Control CActioner Mediator MBroker	DDP Chain of Resp	MBroker logout	Foundation FConnection
R5	PConsole getUserInput PConsole viewMessages MBroker retrieveMessages FReader readMessages FReader readContact	Presentation PConsole Acquaintance IAconstants	DDP Chain of Resp	CActioner retrieveMessages PConsole displayMessages MBroker retrieveContacts MBroker createContacts MBroker createMessages	Mediator MBroker Acquaintance IAEmployee IAOutMessage IAContact EEmployee EOutMessage EContact Foundation FReader entity EContact entity EOutMessage

Mediator MBroker Foundation FReader Acquaintance IAEmployee IAOutMessage IAContact EEmployee EOutMessage EContact	CActioner retrieveMessages MBroker isInCache PConsole displayMessageText	DDP Chain of Resp	Presentation PConsole	PConsole getUserInput PConsole displayMessageText	R6
Foundation FWriter	MBroker updateMessage MBroker flagCache	DDP Chain of Resp Mediator	Presentation PConsole Control CActioner Acquaintance IAconstants Foundation FWriter	PConsole getUserInput PConsole prepareMessage CActioner sendMessage PConsole displayConfirmation FWriter updateMessage	R7
Foundation FConnection	MBroker logout	DDP Chain of Resp	Presentation PConsole Ref. R1 Control CActioner	Ref. R1 PConsole displayLoginError CActioner exit	R8
Foundation FConnection	MBroker logout	DDP Chain of Resp	Presentation PConsole	PConsole getUserInput PConsole displayMenu CActioner logout ref. R1	R9
				Ref. R5 PConsole tooManyMessages	R10
			Presentation PConsole	Ref. R7 CActioner handleEmailException PConsole displayEmailFailure PConsole displayMenu	R11

شكل (٨-٣) توسعة جدول إيجاد الأصناف

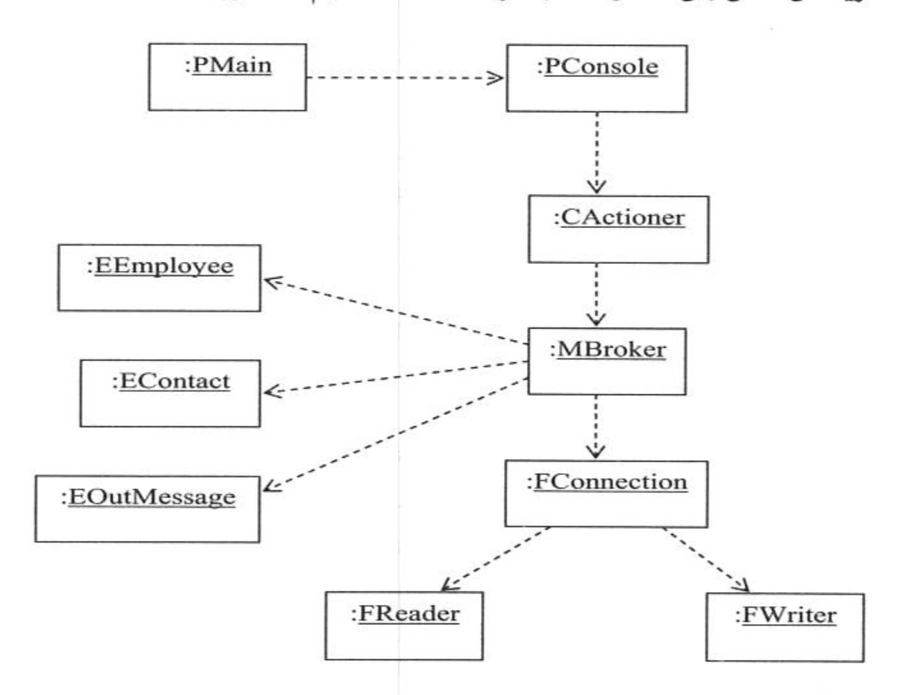
نلاحظ من الشكل أن الجدول يحتوى على ٦ أعمدة، الثلاثة الأولى منها مأخوذة من جدول إيجــاد الأصناف (شــكل (٨-١)) والثلاثة الأخيرة هى نتاج عملية التوســعة، حيث تم إضافة أصناف الكائنات entity classes وتوسعة الحزمة Acquaintance وإضافة عمليات للأصناف الموجودة فى باقى الحزم. يوضح شــكل (٨-٤) النســخة المطــورة لنموذج الأصناف الخاص بالتطبيق EM بعد تطبيق الإضافات والتحسينات المذكورة فى الجدول السابق (شكل (٨-٣)). نلاحــظ من النموذج المطور إضافة حزمة entity إلى حزمة mediator لتكوين حزمة أشــمل domain وهــى تمثل إحدى طبقات الإطــار المعمارى PCMEF. ونلاحظ أيضاً إضافة ثلاثة أصناف واجهة إلى حزمة Acquaintance يتم تطبيقها من خلال أصناف حزمة entity.



شكل (٨-٤) مخطط الأصناف للتطبيق EM

٨-٣ مخطط الكائنات Object Diagram:

يمثــل الكائن إحدى حالات الصنف فى وقت التنفيذ. يتم إنشــاء الكائن الأول فى التطبيق من خلال الطريقة الأساسية للبرنامج main الموجودة غالباً فى حزمة الواجهة. يقوم هذا الكائن بإنشــاء حالات أخرى للأصنــاف المرتبطة به طبقاً للتصميم المعمارى لنموذج الأصناف وذلك من خلال تبادل الرســائل بين الكائنات. يوضح شــكل (٨-٥) مخطط الكائنات للتطبيق EM والمقابل لمخطط الأصناف فى الشــكل السابق. نلاحظ من الشكل أن الرسائل بين الكائنات تتبع علاقات التبعية بين الأصناف وتعبر الطبقات المعمارية من أعلى إلى أسفل حسب مواصفات التصميم المعمارى PCMEF.



شكل (٨-٥) مخطط الكائنات للتطبيق EM

٨-٤ التفاعلات Interactions:

بعد الانتهاء من دراســة النظام فى حالة السكون وتصميم نموذج الأصناف للنظام، ننتقل الآن إلى دراسة النظام وهو فى حالة الحركة أو ما يسمى بدراسة سلوك النظام System Behavior. تركز عملية الســلوك على ملاحظة عمليات تبادل المعلومات بين

كائنـــات النظام فى أثناء تأدية وظيفة معينة وهو ما يطلق عليه اســـم «التفاعلات بين الكائنـــات». يتم تحقيق هذه التفاعلات من خلال مجموعة من الرســـائل Messages المتبادلة بين الكائنات. تعتبر الرسالة هى وسيلة اتصال بين كائن مرسل وآخر مستقبل للرســـالة. تدعم لغة UML2 مجموعة من النماذج التى تســـتخدم لوصف أســـاليب التفاعـــلات بين كائنات النظام وهى مخطط التتابـــع Sequence Diagram ومخطط الاتصال Communication Diagram ومخطط التفاعل Interaction Diagram.

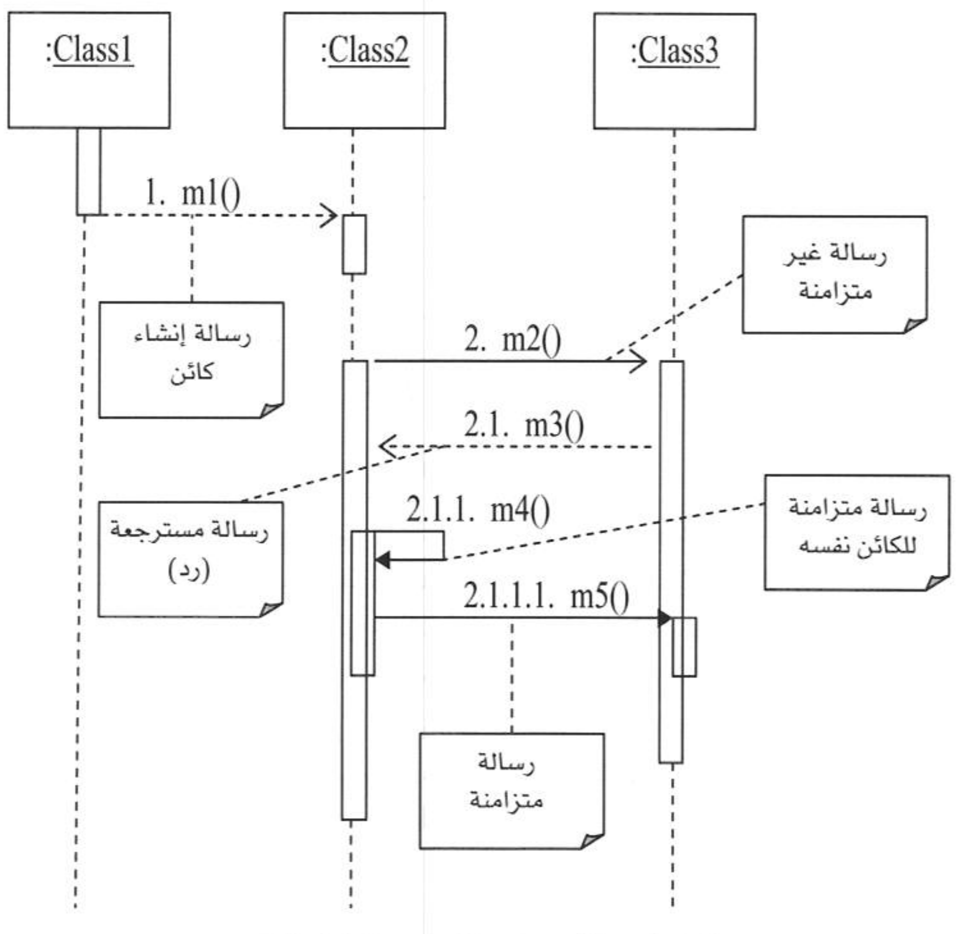
٨-٤-١ مخطط التتابع Sequence Diagram:

يمثل مخطط التتابع مجموعة التفاعلات بين الكائنات داخل حالة اســـتخدام معينة من خلال مجموعة الرسائل الممررة بين الكائنات. يتم عرض الكائنات فى أعلى المخطط ويتم رســـم خط رأســـى لكل كائن يمثل خط الحياة له Life Line. يتم تمثيل الرســـالة المرســـلة من كائن إلى آخر بخط له سهم يتجه نحو الكائن المستقبل للرسالة. يتم رسم مســـتطيل ضيق رأسى حول خط حياة الكائن يمثل فترة تنشيط الكائن فى أثناء عملية تنفيـــذ العملية المطلوبـــة منه. يوضح هذا المخطط تتابع الرســـائل بين الكائنات حيث إنه يهتم بوقت التنفيذ الموضح فى الاتجاه الرأســـى. هناك نوعان من الرســـائل: يمثل النوع الأول الرســـائل المتزامنة Synchronous التى ينتظر فيها الكائن المرسل للرسالة حتى ينتهى الكائن المســـتقبل لها مـــن تنفيذ الإجراءات المطلوبـــة. ويمثل النوع الثانى الرســـائل غير المتزامنة Asynchronous وفيها يرســـل الكائن الرسالة ولا ينتظر حتى ينتهى الكائن المســـتقبل لها من تنفيذها. يوضح شـــكل (٨-٦) أساليب تبادل الرسائل فى مخطط التتابع. يتم تمثيل الوقت فى مخطط التتابع فى الاتجاه الرأسى ويبدأ من أعلى متجهاً إلى أسفل. ويتم تمثيل الرسائل حسب الأسبقية، فتبدأ الرسائل الموجودة أعلى المخطط قبل الرســـائل التى تليها لأســـفل. وعلى هذا فإن مخطط التتابع يعتبر ثنائى الأبعاد، تمثل الكائنات الاتجاه الأفقى وتتجه من اليسار إلى اليمين، ويمثل الوقت الاتجاه الرأسى ويتجه من أعلى إلى أسفل. يلاحظ من الشكل أن أرقام الرسائل تشير إلى ترتيب تنفيذها وأنها ليست ضرورية، حيث إن خط الزمن يوضح عملية التتابع.

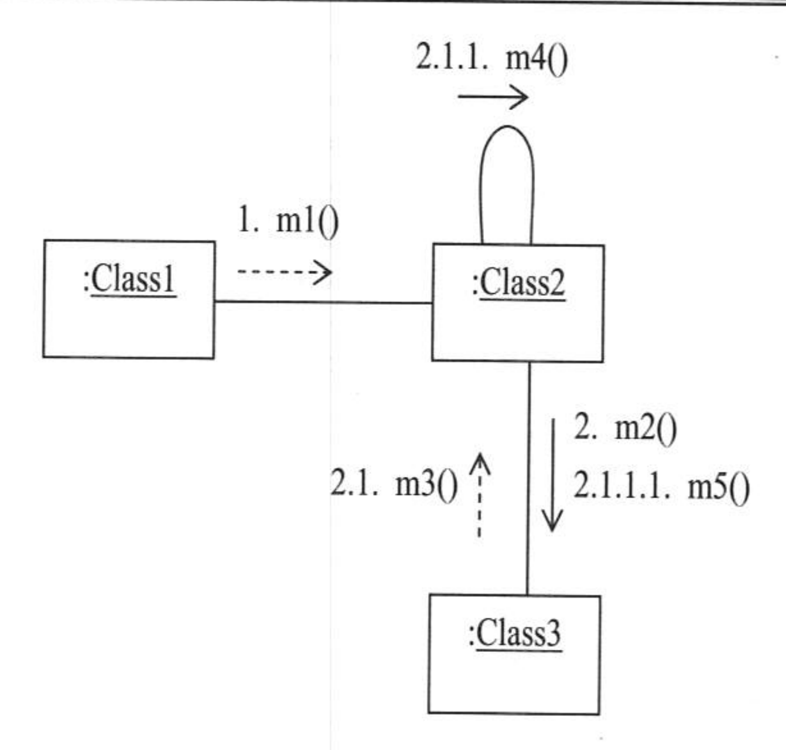
٨-٤-٢ مخطط الاتصال Communication Diagram:

يعتبر مخطط الاتصال هو أحد أشكال مخططات التفاعل Interaction التى تقوم بتمثيـــل التفاعل بين الأشـــياء أو الكائنات. يعتبر مخطط الاتصال شـــبيهاً بمخطط

التتابع من ناحية تمثيل التفاعلات من خلال مجموعة الرسائل المتبادلة بين الكائنات. وجدير بالذكر أن معظم أدوات هندسة البرمجيات Case Tools تقوم بتحويل أحد المخططات إلى الآخر تلقائياً. ويتضح الفارق بينهما من ناحية تركيز مخطط التتابع على التسلسل الزمنى للرسائل بين الكائنات، فى حين يركز مخطط الاتصال على تمثيل العلاقات بين الكائنات بشكل واضح ومفهوم ولذلك فهو يستخدم فى أثناء عملية تحليل الرسائل المرسلة أو المستقبلة لكائن ما، وتبادل وجهات النظر مع المستخدمين وباقى فريق العمل نظراً لسهولة بناء المخطط والتعديل فيه. يوضح شكل (٨-٧) مخطط الاتصال المقابل لمخطط التتابع الموضح بشكل (٨-٦).



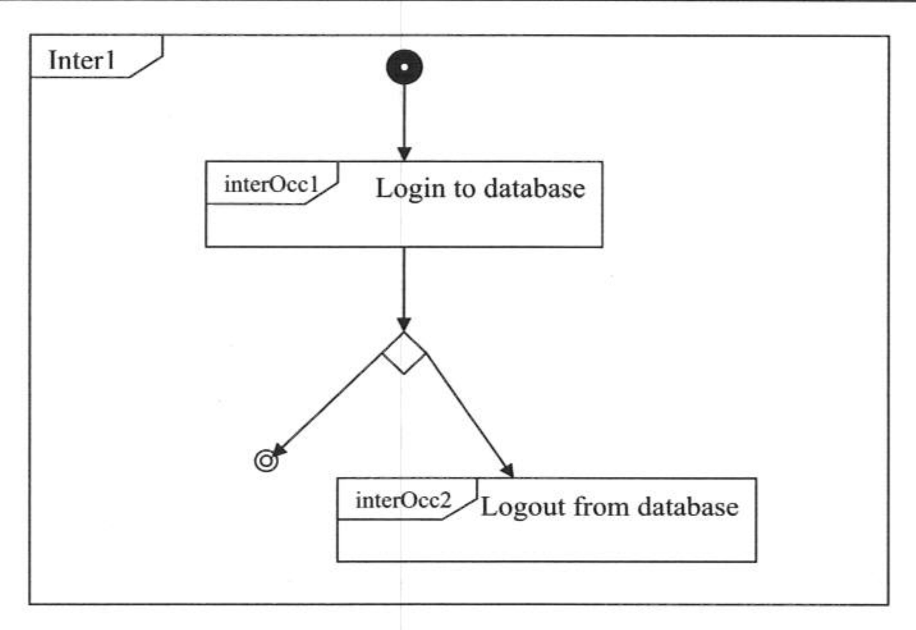
شكل (٨-٦) أنواع الرسائل فى مخطط التتابع



شكل (٨-٧) تبادل الرسائل فى مخطط الاتصال

٨-٤-٣ مخطط التفاعل Interaction Diagram:

تعتبر مخططات التفاعل هى الأســاس فى نمذجة ســلوك النظام باســتخدام لغة UML. تتم عملية نمذجة الســلوك فى عدة مستويات من الإيجاز والتفصيل بدءاً من حالة اســتخدام أو مجموعة من حالات الاســتخدام، إذ تتضمن بعض هذه التفاعلات بداخلها مجموعة من التفاعلات الداخلية. يؤدى ذلك إلى إمكانية الاستعانة بمخطط عام للتفاعلات يطلق عليه اسم «المنظور العام لمخطط التفاعل Interaction Overview Diagram" وهو يشبه إلى حد ما مخطط النشاط Activity diagram. يوضح شكل (٨-٨) مثــالاً على المنظور العام لمخطط التفاعل الذى يمثــل الانتقال بين التفاعلات الداخلية.



شكل (٨-٨) المنظور العام لمخطط التفاعل

٨-٥ تطبيق على التفاعلات Interactions Application:

سنحاول فى هذا القسم تصميم التفاعلات الخاصة بتطبيق إدارة البريد الإلكترونى EM. يحتوى هذا التطبيق على مجموعة محدودة من الأفعال Actions تتضمنها قائمة تشغيل التطبيق، بالإضافة إلى مجموعة الاستثناءات Exceptions أو رسائل الخطأ الموضحة فى مواصفات الاستخدام. وفيما يلى وصف لمجموعة التفاعلات التى سيتم عرضها:

١- تفاعلات أساسية:

- الدخول Login.
- الخروج Exit.
- عرض الرسائل غير المرسلة View Unsent Messages.
- عرض نص الرسالة Display Message Text.
- إرسال الرسالة Email Message.

٢- تفاعلات استثنائية:

- كلمة مرور غير صحيحة Incorrect Username or Password.
- اختيار غير صحيح Incorrect Option.
- عدد كبير جداً من الرسائل Too Many Messages.
- عدم إمكانية إرسال الرسالة Email could not be sent.

جديــر بالذكر أن هذا التطبيق لا يحتاج إلى المنظــور العام لمخطط التفاعل ولكن سيتم اســتخدام مخططات التتابع لوصف التفاعلات الأساسية ومخططات الاتصال لوصف التفاعلات الاستثنائية Exceptions.

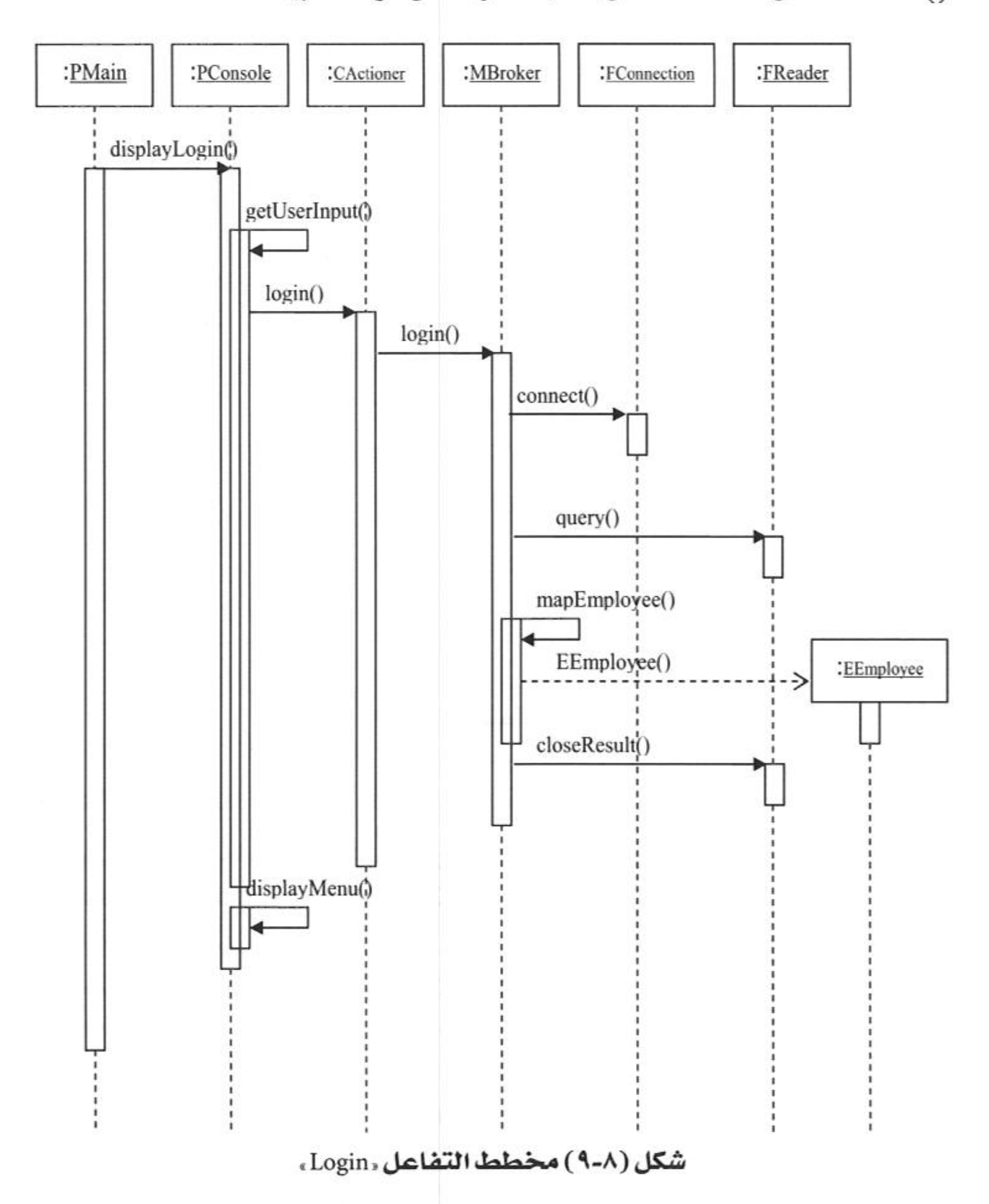
٨-٥-١ التفاعل «Login»:

تبدأ تفاعلات عملية الدخول للتطبيق بالرسالة ( ) displayLogin من الكائن PMain إلـــى الكائن PConsole كما هو موضح فى شــكل (٨-٩). يقــوم الكائن PConsole بعرض الرســالة ( ) getUserInput للمســتخدم للحصول على اسم المستخدم وكلمة المــرور، ثم يقوم بتمريرها من خلال الرســالة ( ) login إلى الكائن CActioner الذى يمررهــا بدوره إلى الكائن MBroker الذى يســتخدمها فى الاتصال بقواعد البيانات ومن ثم تكوين وتمرير الاســتعلام ( ) query إلى الكائــن FReader الذى يقوم بتنفيذ الاســتعلام والبحث عن بيانات الموظف فى قواعد البيانات ويعود بالنتائج إلى الكائن MBroker الــذى يقوم بتعبئة بيانات الكائن Employee، ثم يقوم بإغلاق عملية تنفيذ الاســتعلام. وفى النهاية يقوم الكائن PConsole بعرض قائمة الاختيارات للمستخدم من خلال تنفيذ الطريقة ( ) displayMenu.

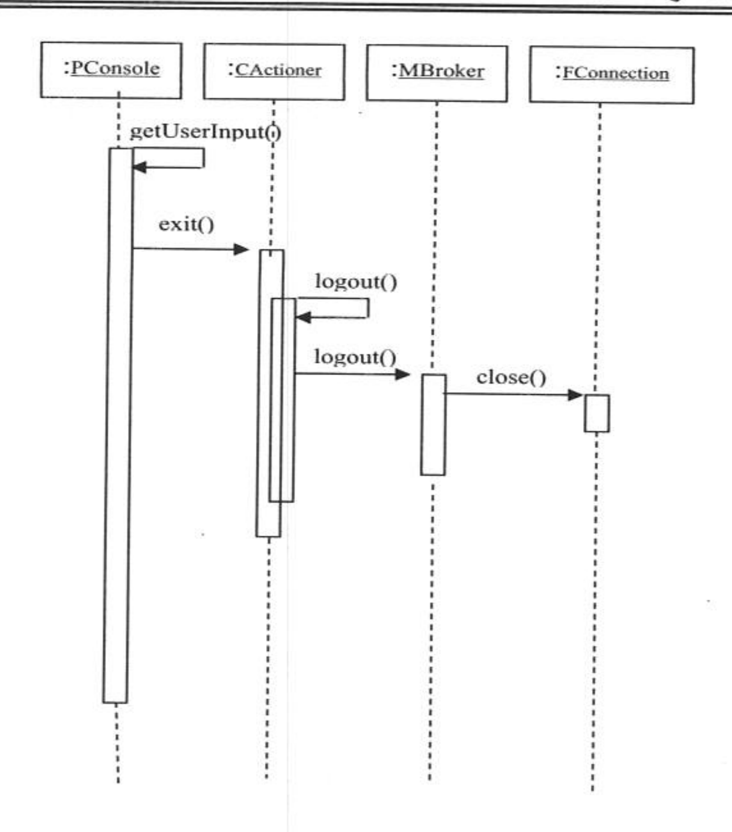
٨-٥-٢ التفاعل «Exit»:

يوضح الشكل (٨-١٠) مجموعة الرسائل التى تمثل التفاعل «Exit» فى التطبيق EM وهى عبارة عن مجموعة من الرسائل الممررة بين الكائنات FConnection، PConsole لغلق قواعد البيانات، وكذلك رسالة إلى الكائن CActioner للخروج من النظام. تبدأ التفاعــلات بتنفيذ الطريقــة ( ) getUserInput للكائن PConsole الذى يقوم بتمرير القيمــة «خــروج» إلى الكائــن CActioner من خلال الرســالة ( ) exit. يقوم الكائن CActioner بتنفيذ الطريقة ( ) logout التى من خلالها يقوم بتمرير الرسالة logout

للكائن FConnection لغلق الاتصال بقواعد البيانات، وهو بدوره يمرر الرســالة () Close للكائن FReader لغلق عملية القراءة من قواعد البيانات.



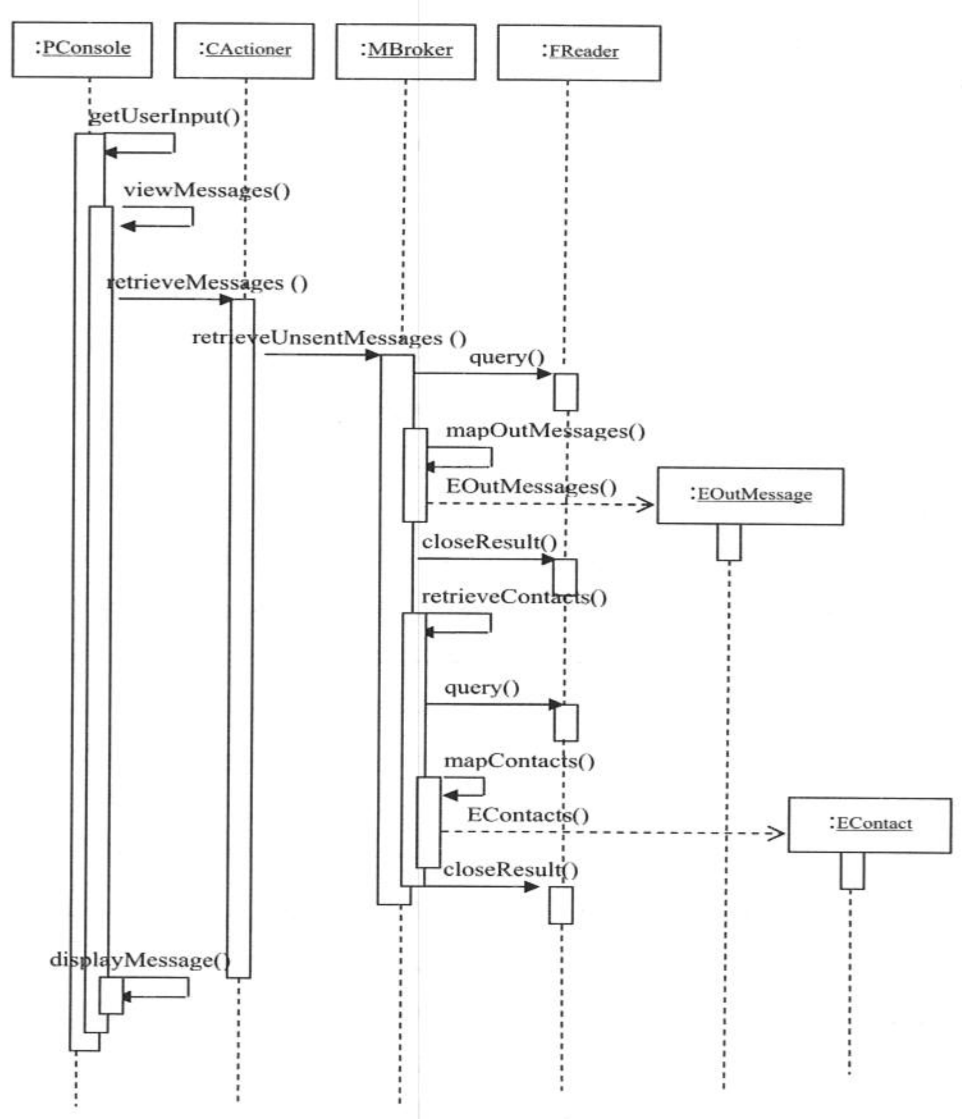
شكل (٨-٩) مخطط التفاعل «Login»



شكل (٨-١٠) مخطط التفاعل «Exit»

٨-٥-٣ التفاعل «View Unsent Messages»:

يبدأ التفاعل بطلب المستخدم عرض الرسائل غير المرسلة من خلال الرسالة viewMessages للكائن PConsole الذى يطلب الخدمة retrieveMessages من الكائن CActioner الذى يمرره بدوره إلى الكائن MBroker مصحوباً بمعلومات الموظف المطلوب عرض رسائله. يقوم الكائن MBroker بطلب تنفيذ الاستعلام من الكائن FReader الذى يقوم بإنشاء كائن جديد EoutMessage ليتضمن مجموعة الرسائل الناتجة من الاستعلام. يتم استرجاع النتائج للكائن MBroker الذى يحتوى على بيانات جهة الاتصال لكل رسالة ويقوم بإنشاء كائنات لكل جهة اتصال EContact. وأخيراً يقوم الكائن PConsole بعرض الرسائل وجهات الاتصال باستخدام الخدمة displayMessage كما هو موضح بالشكل (٨-١١).



شكل (٨-١١) مخطط التفاعل « View Unsent Messages »

٨-٥-٤ التفاعل «Display Message Text»:

مــن المفترض أن يبدأ هذا التفاعل بعد إتمام التفاعل الســابق وفيه يقوم الكائن MBroker بالتحقــق مــن وجود مجموعة الرســائل فى الذاكرة مــن خلال الكائن

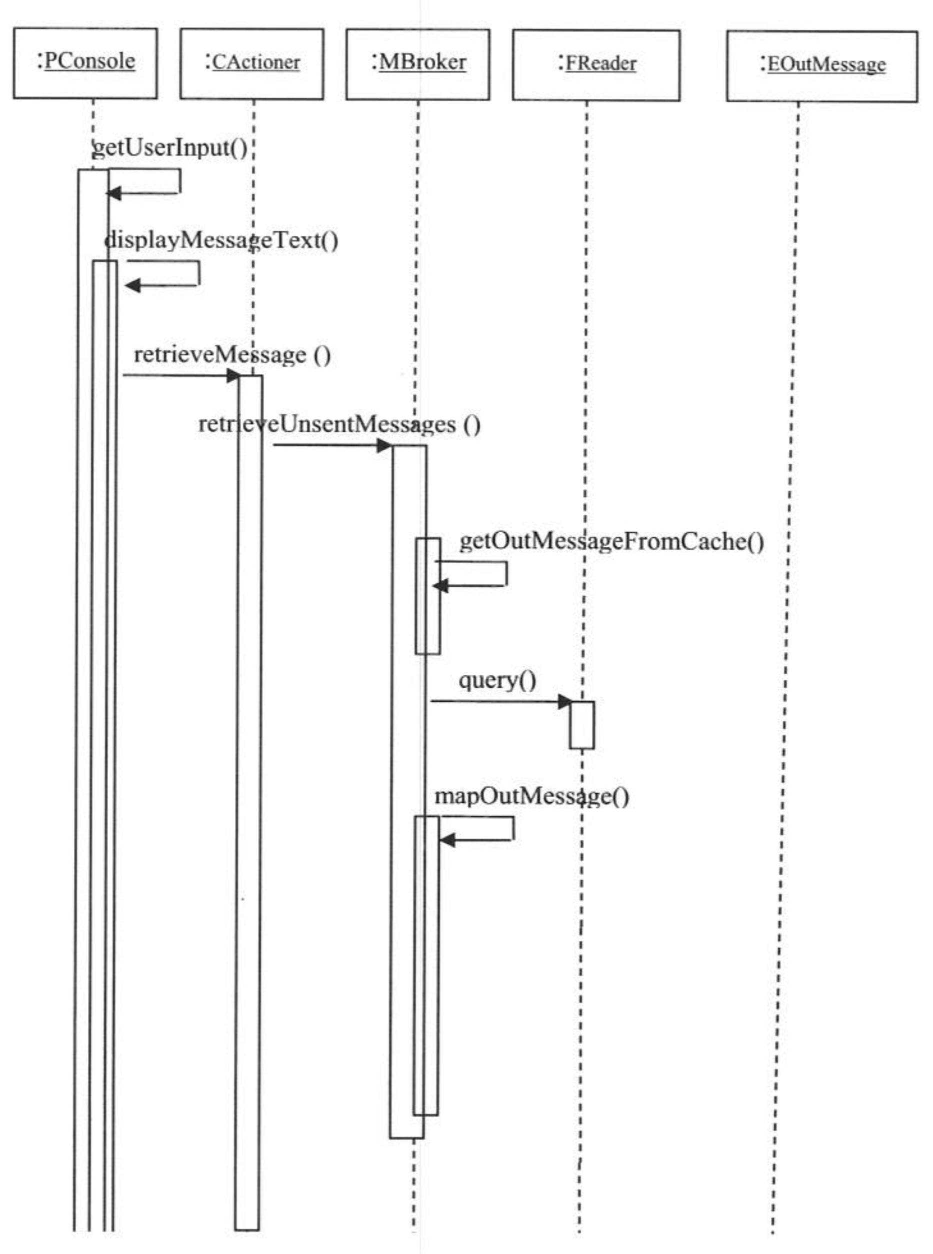
EoutMessage. وإذا لـم يكـن موجـوداً يقوم الكائن بتكوين الاستعلام () query وإرساله إلى الكائن FReader كما سبق شرحه. وبعد ذلك يقوم الكائن MBroker باسترجاع الرسائل إلى الكائن PConsole الذى يقوم بعرض نص الرسالة كما هو موضح بالشكل (٨-١٢).

٨-٥-٥ التفاعل «Email Message»:

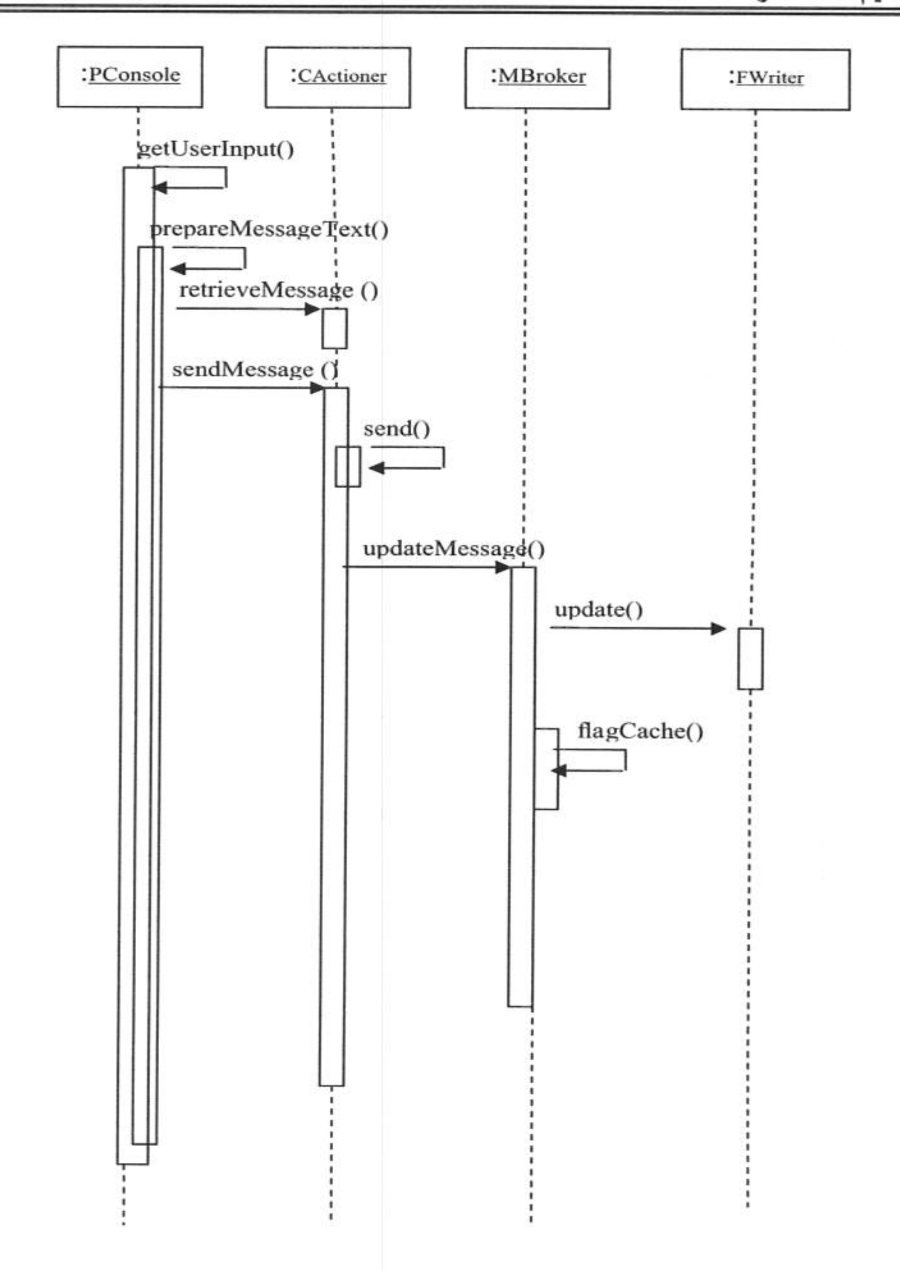
يبدأ التفاعل عندما يختار المستخدم أن يرسل رسالة تحتوى على نص موجود فى قاعدة البيانات. يقوم الكائن PConsole بتجهيز الرسالة باستخدام الطريقة () prepareMessage التى تقوم بإحضار نص الرسالة مستخدماً الخدمة () retrieveMessage كما سبق عرضه فى التفاعل السابق. وبعد ذلك يقوم الكائن PConsole بطلب إرسال الرسالة () sendMessage من الكائن CActioner الذى يقوم باستخدام الطريقة () send. وبعد نجاح عملية الإرسال تقوم الطريقة () sendMessage بتنشيط الطريقة () updateMessage عند الكائن MBroker الذى يمرر جملة SQL إلى الكائن FWriter لتحديث قاعدة البيانات كما هو موضح بالشكل (٨-١٣).

٨-٥-٦ التفاعل «Incorrect User Name or Password»:

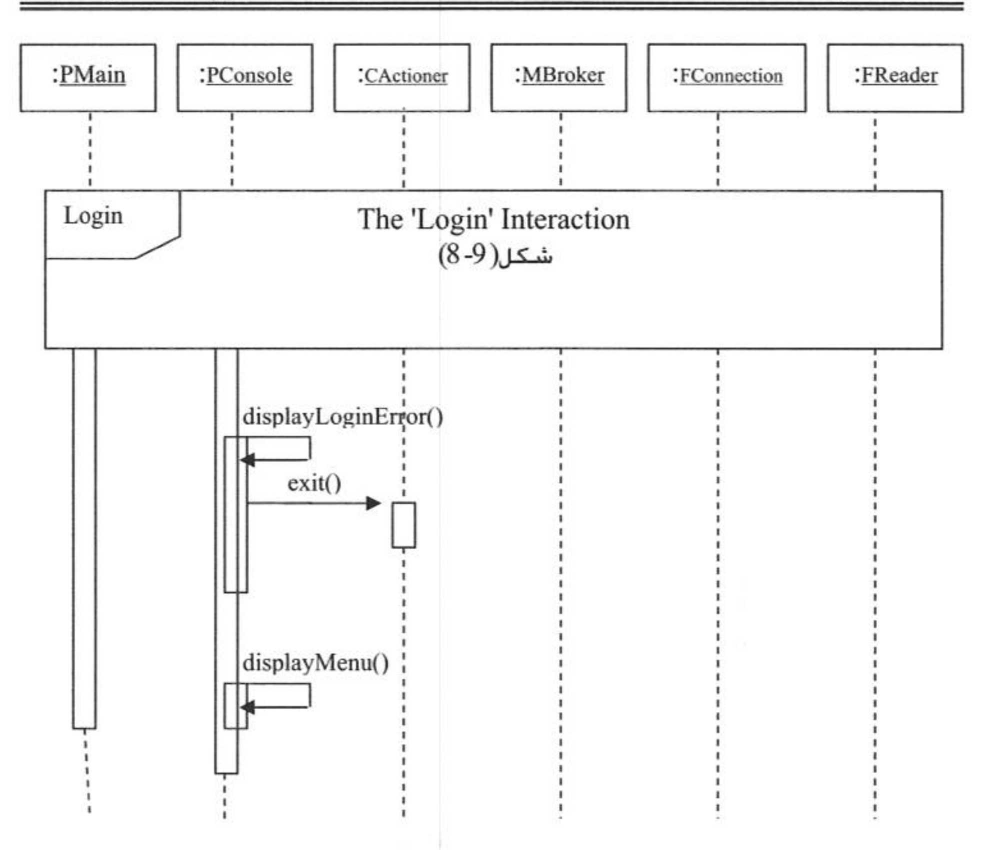
يبدأ هذا التفاعل فى أثناء تنفيذ التفاعل «Login»، فعندما يسترجع الكائن PConsole القيمة «Unsuccessful Login» من تنفيذ الخدمة () getUserInput، عندئذ يستخدم الكائن الطريقة () displayLoginError لإظهار رسالة الخطأ للمستخدم، ثم يستخدم الطريقة () displayMenu لعرض قائمة الاختيارات للمستخدم. إذا كانت هذه هى المرة الثالثة للخطأ فيقوم الكائن PConsole بإرسال الرسالة () exit إلى الكائن CActioner للخروج من النظام، ثم يعرض القائمة الرئيسية displayMenu كما هو موضح بالشكل (٨-١٤).



شكل (٨-١٢) مخطط التفاعل «Display Message Text»



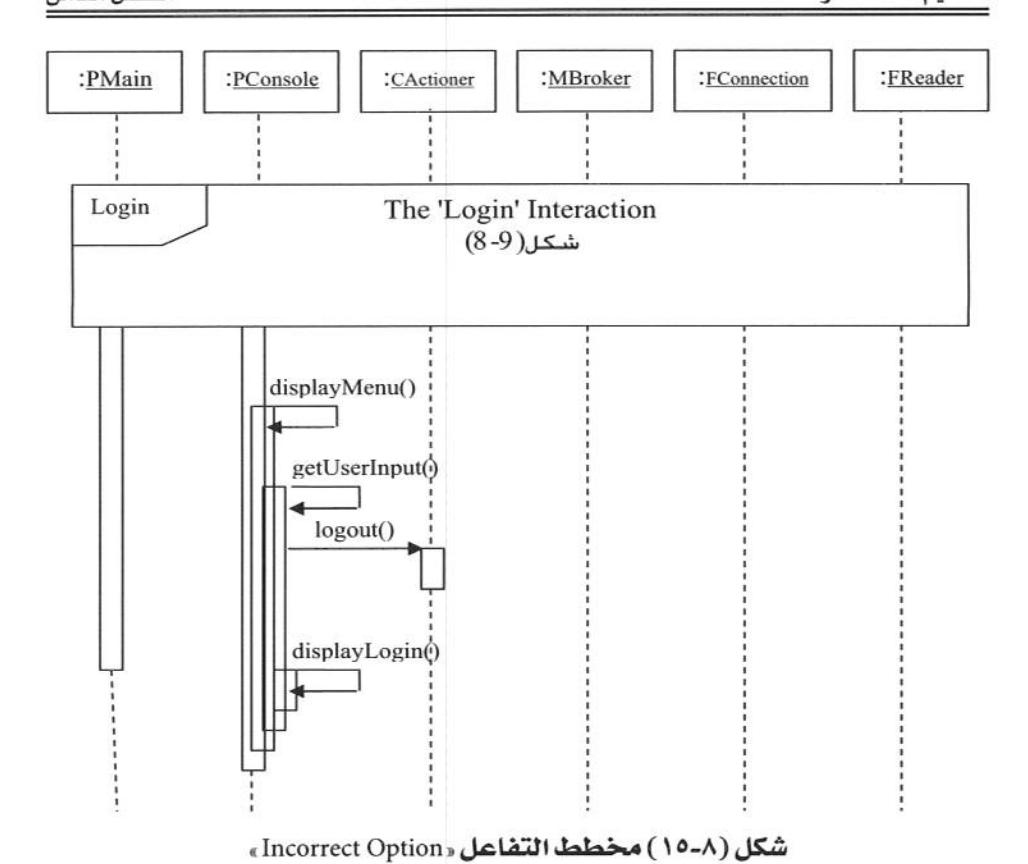
شكل (٨-١٣) مخطط التفاعل « Email Message »



شكل (٨-١٤) مخطط التفاعل «Incorrect User Name or Password»

٨-٥-٧ التفاعل «Incorrect Option»:

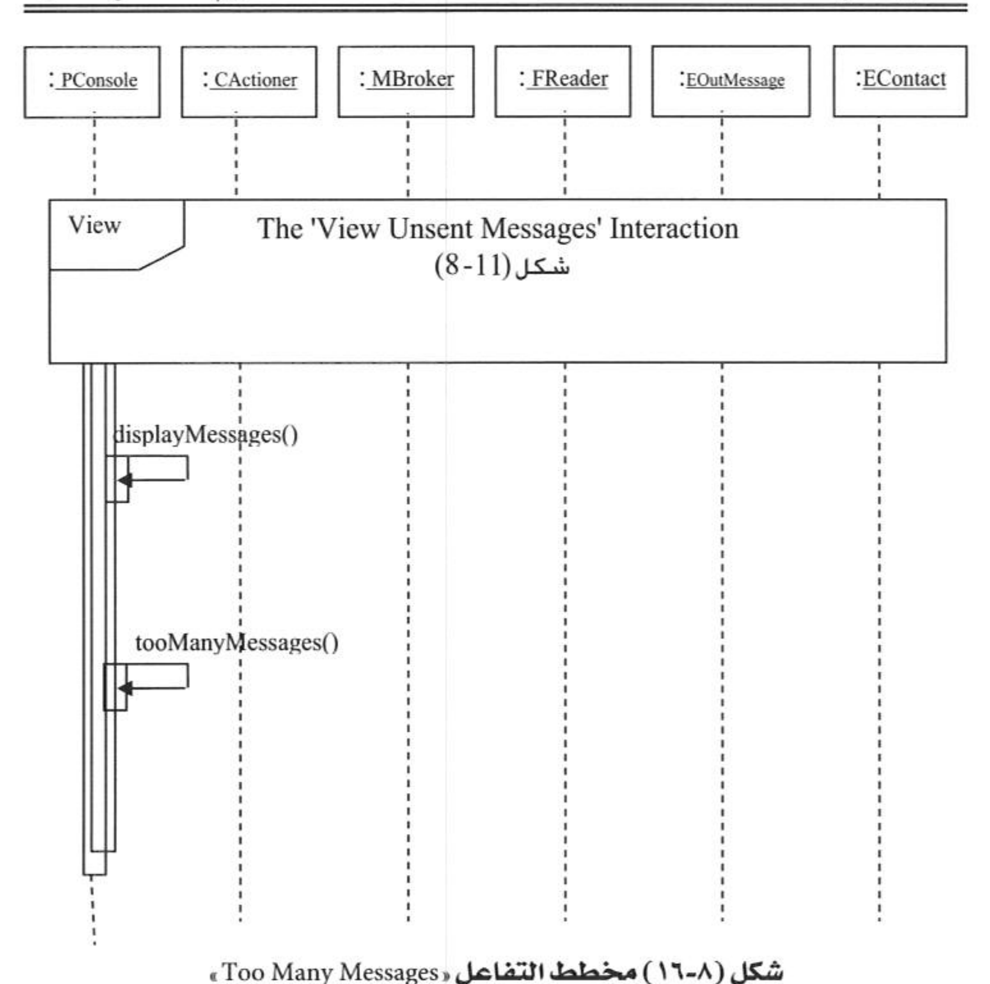
فى حالة نجاح عملية الدخول «Login» يقوم الكائن PConsole باستخدام الطريقة () displayMenu لعرض اختيارات القائمة للمستخدم. وبعد ذلك يستخدم الطريقة () getUserInput لقراءة رقم الاختيار المدخل من المستخدم. إذا كان الاختيار المدخل غير صحيح يتم إعادة عرض قائمة الاختيارات للمستخدم مرة أخرى. وفى حالة تكرار الخطأ ثلاث مرات متتالية يقوم الكائن PConsole باستدعاء الطريقة () logout، ثم تعرض شاشة "Login" مرة أخرى، كما هو موضح بالشكل (٨-١٥).



شكل (٨-١٥) مخطط التفاعل «Incorrect Option»

٨-٥-٨ التفاعل «Too Many Messages»:

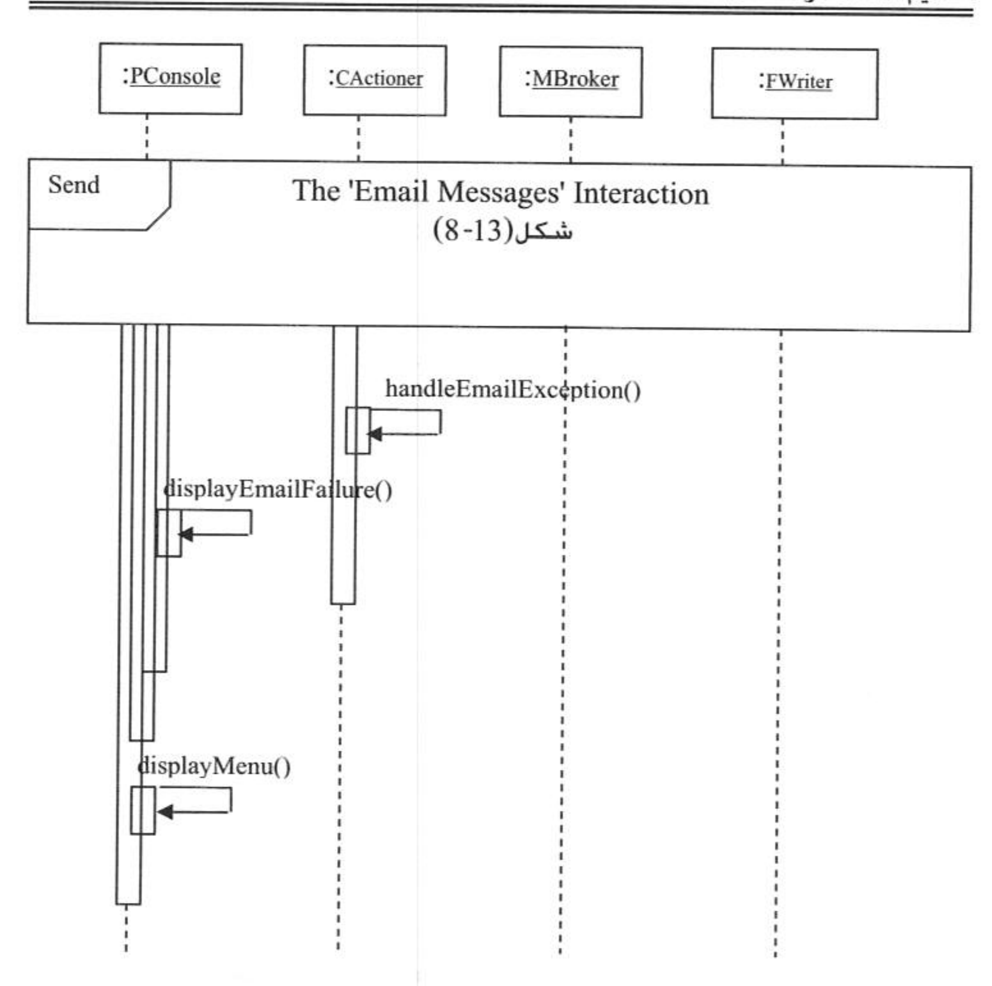
يســتخدم الثابت Max - Display - Messages الموجود فى الصنف IAContacts (شــكل (٨-٤)) للتحكم فى أقصى عدد للرسائل التى يمكن أن يعرضها البرنامج على شاشة المســتخدم. يعتبر هذا التفاعل امتداداً للتفاعل «View Unsent Messages»، إذ يتم اســتدعاء الطريقة () retrieveMessages مصحوبة بمعامل يمثل عدد الرسائل numMsgs ويتــم مقارنتــه بالثابــت Max ـ Display ـ Messages. إذا كان العــدد numMsgs أكبر من المسموح به فيتم عرض عدد الرسائل المسموح بها فقط باستخدام الطريقــة () displayMessages ويتبعها تنفيذ الطريقة tooManyMessages لإعلام المســتخدم بوجود عدد آخر من الرســائل المتبقية فى قاعدة البيانات ولم يتم عرضها كما هو موضح بالشكل (٨-١٦).



شكل (٨-١٦) مخطط التفاعل «Too Many Messages»

٨-٥-٩ **التفاعل** «Email Could Not be Sent»:

يبدأ هــذا التفاعل فــى أثنــاء تنفيــذ التفاعــل «Email Message» عند حدوث اســتثناء فــى أثناء عمليــة الإرســال نتيجة عــدم إتمامها بنجــاح، ويتم اســتدعاء الطريقــة handleEmailException، ويقــوم الكائن PConsole باســتدعاء الطريقة displayEmailFailure لإعلام المســتخدم بفشل عملية إرســال الرسالة. وبعد ذلك يقوم بإعادة عرض قائمة الاختيارات كما هو موضح بالشــكل (٨-١٧)، مع عدم تنفيذ التفاعل الخاص بتحديث قاعدة البيانات.



شكل (٨-١٧) مخطط التفاعل «Email Could Not be Sent»

الفصل التاسع
البرمجة والاختبار
Programming and Testing

من المعروف أن الغرض الأساسى من عملية تطوير النظم هو إنتاج برمجيات قادرة على تحقيق متطلبات المستخدم. تعتبر عمليات البرمجة والاختبار هى المسؤولة عن إنتاج تلك البرمجيات حسب المواصفات التى تم تحديدها فى أثناء تحليل المتطلبات وتصميم الحلول المناسبة لها. يتم اختبار البرمجيات المنتجة للتأكد من مطابقتها لمواصفات المتطلبات وبالجودة المطلوبة. ومن ثم فإن عمليات الاختبار تلازم عمليات البرمجة وأحياناً تكون عمليات البرمجة تابعة لعمليات اختبار محددة Test-Driven Programming. يقوم فريق المبرمجين بتطوير برامج الاختبار لكل وحدة برمجية منتجة، وهى عملية تتطلب وقتاً وجهداً كبيرين وتحتاج إلى تكلفة كبيرة. وفى النهاية يتم إجراء اختبار القبول Acceptance Test للنظام للتأكد من تحقيقه لجميع متطلبات المستخدم. وجدير بالذكر أنه يتوافر حالياً مجموعة كبيرة من لغات البرمجة وأدوات هندسة البرمجيات CASE Tools التى تستخدم فى برمجة واختبار النظم البرمجية. يعود الاختلاف بين هذه اللغات والأدوات إلى الأساليب والمنهجيات التى تدعمها فى فى أثناء عمليات التطوير. وسوف نقدم الآن فكرة عامة عن لغة جافا JAVA التى تعتبر من أكثر اللغات استخداماً فى تطوير التطبيقات الحديثة نظراً لدعمها لأسلوب البرمجة المبنية على الكائنات Object-Oriented Programming الذى يلائم أسلوب تحليل وتصميم النظم باستخدام لغة UML، بالإضافة إلى دعمها القوى لتقنية برمجة التطبيقات التى تعمل على الويب.

٩-١ فكرة عامة عن لغة جافا Overview of JAVA Language:

سنحاول فى هذا الجزء تقديم عرض سريع للسمات الأساسية للغة جافا وذلك من وجهة نظر هندسة البرمجيات. يركز العرض على المفاهيم الأساسية للغة جافا مثل الأصناف والعلاقات بينها والاتصال بقواعد البيانات التى تستخدم فى تطوير معظم التطبيقات. وسوف نقوم بالاستعانة بالتطبيق الخاص بالتعامل مع قاعدة بيانات الأفلام Movie Actors التى تم تصميمها فى الفصل السابع من الكتاب وذلك لتوضيح المفاهيم الأساسية للغة جافا.

٩-١-١ الأصناف Classes:

يمثل الصنف فى لغة جافا القالب الذى يتم إنشاء كائنات جديدة منه. وهو يحتوى على مجموعة من المحددات التى تصف المفاهيم التالية:

- اسم الصنف ومجال استخدامه Visibility.
- بيانات الكائن Data Members.
- العمليات التى يقوم بها الكائن Methods.
- الوحدات البنائية Constructors.
- مجال استخدام أعضاء الكائن (البيانات والعمليات).
- الصنف العلوى له Super class (إن وجد).
- واجهات الصنف Interfaces (إن وجدت).

تحدد خاصية Visibility مجال استخدام كل من كائنات وخصائص الصنف. غالباً ما يتم تعريف مجــال الصنف على أنه عام Public لتتمكن باقى الكائنات من التعامل مع حالات الصنف Class Instances. ويتم تعريف معظم بيانات الصنف Class Data Members علــى أنها خاصة Private حتى يقتصر التعامــل معها من خلال عمليات الصنف نفســه، فى حــين يتم تعريف معظم عمليات الصنف علــى أنها عامة Public حتى يتــم التعامل معها من خلال كائنات أصناف أخرى. يتم تعريف كل من متغيرات الصنف Class Variables وطرق الصنف Class Methods باستخدام الكلمة المحجوزة Static.

تدعم لغة جافا أســلوب التحميل الزائد للطرق Method Overloading، إذ تسمح لأكثر من طريقة داخل الصنف باســتخدام الاســم نفســه ولكن مع اختلاف فى عدد المتغيرات الممررة Parameters للطريقة ونوعها. تدعم لغة جافا أيضاً أسلوب «نموذج الطريقــة Method Prototype" الذى يتكون من اســم الطريقة والمعاملات الخاصة بها وأنواعها ونوع الناتج للطريقة ولكن بدون محتوى. يســتخدم هذا النوع من الطرق لتعريف أصناف الواجهة Interface Classes.

يحتـوى الصنـف على نوع خـاص من الطـرق يسـمى Constructor وهو يأخذ اسـم الصنف نفسه ويستخدم فى إنشـاء حالات الصنف (الكائنات الجديدة للصنف Objects). يمكـن أن تأخذ هذه الطريقة عدة أشـكال Overloading وذلك لإنشـاء كائنـات جديدة بطرق مختلفة. يوضح شـكل (٩-١) تصميم الصنف Movie المقابل لجدول الأفلام. يحتوى الصنف على ثلاثة متغيرات Data Members، وطريقة إنشاء الكائنـات Movie Constructor، ومجموعة من الطرق Methods التى تقوم بتخزين واسترجاع بيانات الكائن () Get و () Set.

Movie
movieTitle :String movieCode :double director :String
addListedAs(L: ListedAs): void removeListedAs(L: ListedAs): void getMovieTitle(): String getMovieCode(): double getDirector(): String setMovieTitle(): String setMovieCode(): double setDirector(): String equals(o:Object): boolean

شكل (٩-١) تصميم الصنف Movie

يتم اسـتخدام هذا التصميم فى تطبيق الصنف باستخدام لغة جافا. يوضح شكل (٩-٢) محتوى البرنامج الذى يقوم بتطبيق الصنف Movie.

```
Movie.java

Public class Movie {
    Private String  movieTitle;
    Private double movieCode;
    Private String  director;
    Private Collection ListedAs;

    Public Movie(double movieCode, String title, String director)  {
            This.movieCode = movieCode;
            This.director = director;
            This.movieTitle = title;
            ListedAs = new ArrayList();
    }

   Public void  addListedAs(ListedAs  l) {
            ListedAs.add(l);
   }

   Public void  removeListedAs(ListedAs  l) {
            ListedAs.remove(l);
   }

   Public String  getMovieTitle() {
            Return  movieTitle;
   }

   Public void  setMovieTitle(String  title)  {
       This.movieTitle = title;
   }

    Public double  getMovieCode()  {
            Return  movieCode;
    }

    Public void  setMovieCode(double  code)  {
            This.movieCode = code;
    }

    Public String  getDirector()  {
            Return  director;
    }

    Public void  setDirector(String  director)  {
            This.director = director;
    }

    Public Boolean  equals(Object  o)  {
            Try {
                    Movie  m = (Movie)  o ;
                    If( m.movieCode == movieCode )  return  true;
                  } catch(Exception  exc)  {}
                   Return  false;
    }
}
```

شكل (٩-٢) محتوى البرنامج Movie.java

٩-١-٢ ارتباط الأصناف Class Associations:

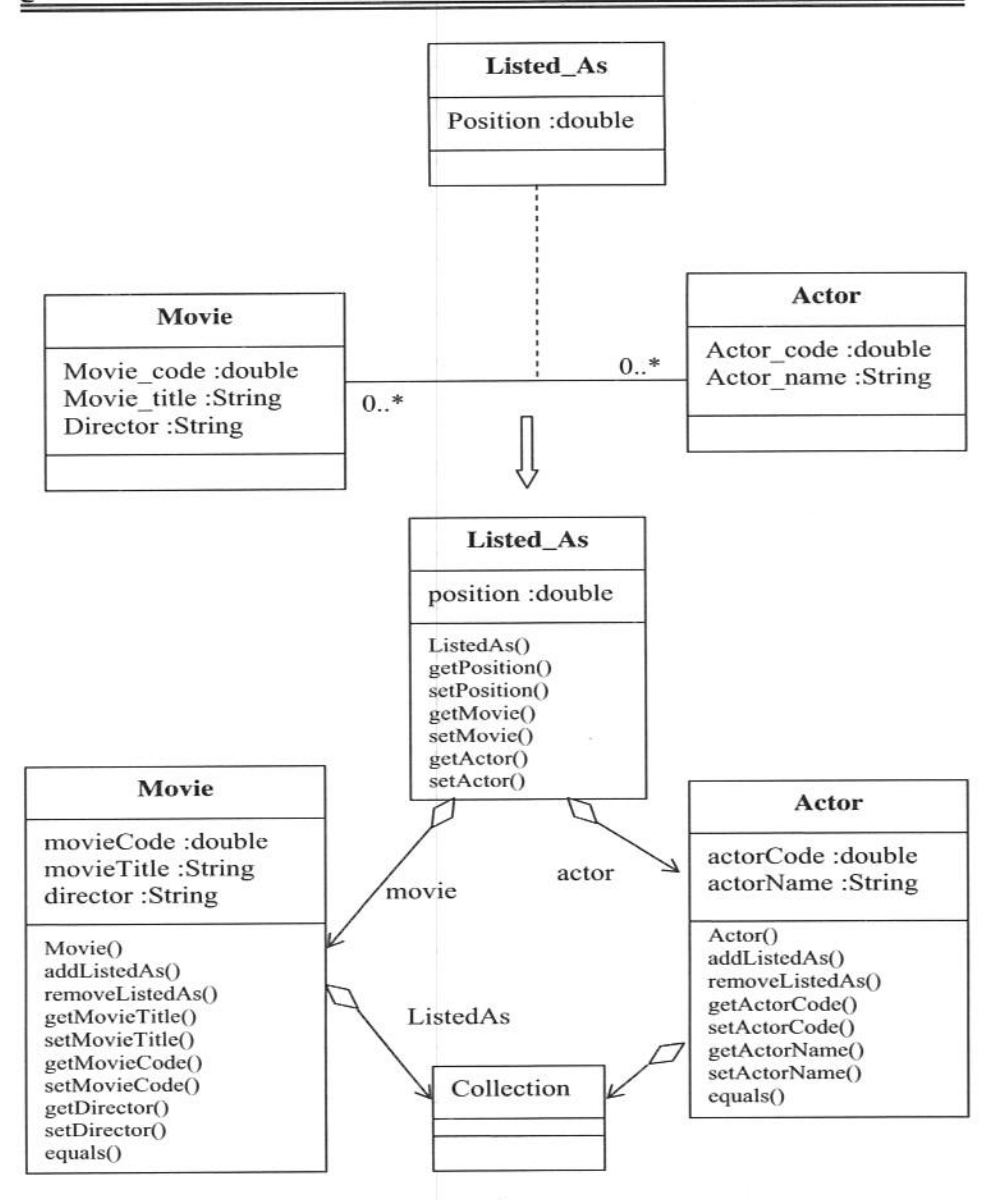
يجب الاهتمام بتعريف مســارات الاتصال بين الكائنــات بوضوح تام فى البرنامج التطبيقى مما يؤدى إلى فهم البرنامج وســهولة صيانته وتوســعته فيما بعد. تســهم

عمليـــة الاتصال بين الأصناف من خلال الرســـائل فى توضيـــح العلاقات بينها. يتم تمثيـــل الارتباط بين الأصناف من خلال متغيرات الحالات Instance Variables التى تخزن معلومات تشير إلى الكائنات المرتبطة. يتم استخدام متغير حالة واحد فى حالة الارتباط بكائن واحد فقط، ويتم اســـتخدام مجموعة متغيرات تســـمى «Collection» فى حالة الارتباط بأكثر من كائن. ولتوضيح ذلك ســـوف نســـتعين بنموذج الأصناف فى تطبيـــق الأفلام يصف علاقة الارتباط من النـــوع متعدد-إلى-متعدد بين الصنف «Actor» والصنـــف «Movie» عن طريق صنف ثالث (صنـــف الارتباط Association Class) وهو صنف «Listed_As» كما هو موضح بالشكل (٩-٣).

ويوضح الشـــكل أيضاً كيفية تحويل هذا النموذج إلى النموذج الذى ســـيتم تطبيقه باســـتخدام لغة جافا. يحتوى النموذج التطبيقـــى على ثلاثة أصناف تمثل كائنات كل مـــن «Listed-As»، «Actor»، «Movie»، ويحتـــوى كل صنف علـــى جميع الخصائص Attributes والطرق Methods الخاصة به. نلاحظ من الشكل أيضاً أنه لم يتم تمثيل العلاقة بين Actor» و«Movie» من خلال علاقتى ربط من النوع واحد - إلى - متعدد كمـــا هو الحال فى مخطـــط الأصناف، ولكن تم تمثيل العلاقة من خلال أربع علاقات ربـــط أحادية الاتجاه وأحاديـــة التعددية ومتضمنة أســـلوب المجموعات Collections الخاص بلغة جافا الذى سنوضحه الآن بشىء من التفصيل.

مجموعات جافا Java Collections:

على النقيض من قواعد البيانات المعتمدة على الكائنات، فإن لغات البرمجة المعتمدة على الكائنات لا تدعم بشـــكل مباشر عملية تطبيق علاقات الربط Associations بين الكائنـــات. ولكنها تعود إلى مهارة المبرمج فى تمثيل العلاقات بين الكائنات من خلال مجموعـــة من الأصناف التى تتضمن معلومات عن علاقـــات الربط المطلوبة. وتزداد هذه العملية صعوبة فى حالة العلاقات ثنائية الاتجاه Bidirectional Associations. ففـــى حالة العلاقات أحادية الاتجاه التى يشـــارك فيها كائـــن واحد فقط يتم تمثيل العلاقة من خلال متغير حالة Instance Variable يأخذ نوع الصنف، وتكون قيمة هذا المتغير فى شكل إشارة Reference إلى كائن من هذا الصنف.



شكل (٩-٣) تحويل نموذج الأصناف إلى النموذج التطبيقى

وفـــى حالة مشـــاركة أكثر من كائن واحد فى العلاقة يتـــم تمثيل العلاقة من خلال متغير حالة يأخذ نوع واجهة مجموعة Collection Interface مأخوذة من مكتبة جافا

java.util.collection. وتكون قيمة المتغير فى شــكل إشــارة إلى بعض الكائنات التى تطبق هذه الواجهة. يوضح شكل (٩-٤) واجهة مجموعة Collection Interface.

Collection
add(obj: Object): Boolean addAll(collection: Collection): Boolean clear(): void contains(obj: Object): Boolean containsAll(collection: Collection): Boolean equals(obj: Object): Boolean hashCode(): int isEmpty(): Boolean iterator(): iterator remove(obj: Object): Boolean removeAll(collection: Collection): Boolean retainAll(collection: Collection): Boolean size(): int toArray(): Object[] toArray(aobj[]: Object): Object[]

شكل (٩-٤) واجهة مجموعة Collection Interface

تعتبــر واجهة المجموعة بمنزلة مجمع للكائنات وهــو يتضمن مجموعة من الطرق التى تسمح لها بإضافة كائنات جديدة أو حذف كائنات منها وكذلك البحث عن وجود كائــن معين والتحقــق من عدد الكائنات الموجودة بالمجمــع. تحتوى مكتبة مجموعات جافــا java.util.collection على ثلاث مجموعــات هى List و Set و Map. يوضح شــكل (٩-٥) برنامجاً بلغة جافا لتمثيل الصنــف Actor الذى يتضمن تعريفاً للمتغير ListedAs من النوع مجموعة Collection الذى يشير إلى مجموعة كائنات ListedAs الموضحة فى شكل (٩-٣). يتضمن البرنامج أيضاً الطريقة () AddListedAs لإضافة كائنات جديدة للمجموعة وطريقة () RemoveListedAs لحذف كائنات من المجموعة ListedAs. ويوضح شــكل (٩-٦) برنامج جافا لتمثيل الصنف ListedAs الذى يبين كيفية قيام الكائن ListedAs بتأســيس العلاقة بــين الكائن Actor والكائن Movie. يتم استخدام إشارة إلى الكائنين فى شكل معاملات تمرر إلى الطريقة البنائية للكائن

ListedAs. وهنــا يقوم الكائن ListedAs بإضافة نفســه كمعامل يمرر إلى الطريقة () AddListedAs لــكل مــن الكائن Movie والكائن Actor، ممــا يؤدى إلى ضمان السلامة المرجعية بين الكائن ListedAs والكائنين Actor و Actor.

Actor.java

```
Public class Actor {

    Private double actorCode;

    Private String actorName;

    Private Collection listedAs;

    Public Actor(double code, String name) {
            this.actorCode = code;
            this.actorName = name;
            listedAs = new ArrayList();
     }

    Public void addListedAs(ListedAs l) {
            listedAs.add(l);
    }

    Public void removeListedAs(ListedAs l) {
            listedAs.remove(l);
    }

    Public double getActorCode() {
          Return actorCode;
    }

    Public void setActorCode(double code) {
          This.actorCode = code;
    }

    Public String getActorName() {
          Return actorName;
    }


    Public void setActorName (String actor) {
          This.actorName = actor;
    }

    Public Boolean equals(Object o) {
          Try {
                Actor a = (Actor) o ;
                If(a.actorCode == actorCode ) return true;
               } catch(Exception exc) {}
               Return false;
    }

}
```

شكل (٩-٥) تمثيل الصنف Actor بلغة جافا

ListedAs.java

```
Public class ListedAs {

    Private double position;

    Private Movie movie;

    Private Actor actor;

    Public ListedAs(Movie m, Actor a, double position) {
        this.movie = m;
        this.actor = a;
        this.position = position;

        movie.addListedAs(this);
        actor.addListedAs(this);
    }

}
```

شكل (٩-٦) تمثيل الصنف ListedAs **بلغة جافا**

٩-١-٣ الاتصال بقواعد البيانات Database Connections:

يتم التعامل مع قواعد البيانات المرتبطة Relational Database باستخدام تعليمات لغة الاستفسار البنائية SQL. تحتوى معظم لغات البرمجة على مكونات خاصة بالاتصال والتعامل مع قواعد البيانات. تتضمن هذه المكونات داخلها تعليمات لغة الاستفسار البنائية. فعلى سبيل المثال نجد أن بيئة التطوير الخاصة بلغة جافا تحتوى على المكون Java Database Connectivity (JDBC) الذى يمكن لغة جافا من الاتصال بقواعد البيانات والتعامل معها من خلال تعليمات SQL. تحتوى لغة جافا على مكونات أخرى تستخدم للتعامل مع قواعد البيانات مثل:

- Enterprise Java Beans (EJB).
- Business Components for Java (BC4J).

تدعم معظم أنظمة إدارة قواعد البيانات DBMS مثل «أوراكل» إمكانية أن تقوم لغات البرمجة مثل «جافا» بتخزين بعض الإجراءات فى قواعد البيانات للتعامل السريع معها وذلك من خلال مجموعة من الإجراءات المخزنة Stored Procedures. يمثل الشكل (٩-٧) جزءاً من برنامج جافا للتعامل مع قواعد البيانات باستخدام تقنية JDBC. تتم عملية الاتصال من خلال الصنف connection والصنف DriverManager من مكتبة Java.SQL. يتم إنشاء كائن الاتصال conn الذى يقوم باستدعاء الطريقة getConnection () مستخدماً ثلاث معاملات تمثل مسار قاعدة البيانات المطلوب الاتصال بها، واسم مستخدم قاعدة البيانات، وكلمة السر للاتصال بقاعدة البيانات.

Connection to the Database

```
Public class Connection {

    Private java.sql.connection conn;

    Public connection () throws Exception {

        Class.forName("oracle.jdbc.driver.oracleDriver");
            Conn = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:oracle8i", "pse1", "pse1");
      }
}
```

شكل (٩-٧) الاتصال بقواعد البيانات فى لغة جافا

يحتوى صنف واجهة الاتصال Connection Interface على ثلاثة طرق تستخدم لتنفيذ تعليمات SQL هى: Statement، preparedStatement، callableStatement.

تستخدم Statement لتنفيذ تعليمات SQL البسيطة التى لا تحتوى على معاملات. وتستخدم prepared Statement لتنفيذ تعليمات SQL المتضمنة شروطاً أو معاملات للبحث فى قواعد البيانات. وتستخدم callableStatement لاستدعاء وتنفيذ الإجراءات المخزنة فى قواعد البيانات Stored Procedures.

ولتطبيــق ذلك علــى قاعدة بيانات الأفلام فإنه يتطلب الاتصــال بقاعدة البيانات والبحث فيها عن بيانات الكائنات وتحميلها فى ذاكرة التطبيق. يوضح شــكل (٩-٨) كائن البحث MovieSearcher الذى يقوم بإنشــاء كائن اتصال Conn وبعد ذلك يقوم باستدعاء الطريقة () readAll. ويوضح الشكل (٩-٩) تفاصيل الطريقة () readAll، حيث يتم تكوين تعليمة SQL وتمريرها إلى الطريقة () query. وبعد ذلك يتم إنشــاء الكائن st الذى يقوم بتنفيذ تعليمة SQL على قاعدة البيانات ويتم تخزين نتائج البحث فى الكائن rs.

MovieSearcher.java

```
Public class MovieSearcher {

    Private Connection conn;
    Private Collection ListedAs;

    Public MovieSearcher() throws Exception {
          Conn = new Connection();
          readAll();
        }

   Public static void main(String args[]) {
            MovieSearcher searcher = new MovieSearcher();
       }
}
```

شكل (٩-٨) برنامج البحث بلغة جافا

```
Connection.java
  Public class Connection {

       Private java.sql.connection conn;

       Public connection () throws Exception {

            Class.forName("oracle.jdbc.driver.oracleDriver");
                 Conn = DriverManager.getConnection(
                      "jdbc:oracle:thin:@localhost:١٥٢١:oracle٨i", "pse١", "pse١");
       }
       Public Collection readAll() throws Exception {
            Collection c = new ArrayList();
            ResultSet rs = query("Select * from movie m, actor a, listed_as l
                                 Where m.movie_code = l.movie_code
                                 And l.actor_code = a.actor_code");
            While (rs.next()) {
                  Double movieCode = rs.getDouble("movie_code");
                  String movieTitle = rs.getString("movie_title");
                  String director    = rs.getString("director");
                  Double actorCode  = rs.getDouble("actor_code");
                  String actorName = rs.getString("actor_name");
                  Double position    = rs.getDouble("position");

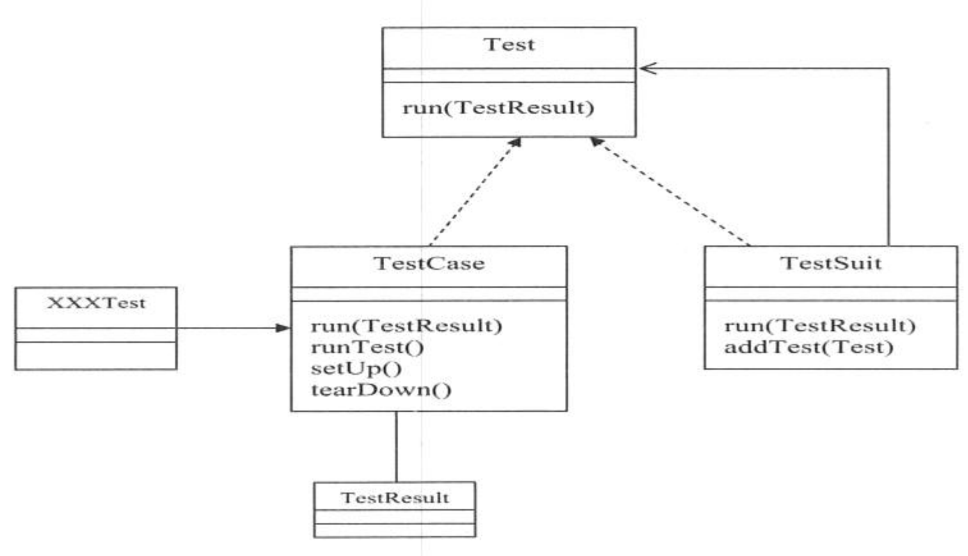
                 Movie m = new Movie(movieCode, movieTitle, director);
                 Actor a  = new Actor(actorCode, actorName);
                 ListedAs la = new ListedAs(m, a, position);
                 c.add(la);
            }
            Return c;
       }
       Private ResultSet query(String sql) throws Exception {
            Statement st = conn.createStatement();
            Return st.executeQuery(sql);
       }
  }
```

شكل (٩-٩) برنامج الاتصال وتنفيذ عملية البحث فى قواعد البيانات

٩-٢ التطوير المبنى على الاختبار Test-Driven Development:

تقــوم بعض أســاليب تطوير البرمجيات على فكرة الاختبار بشــكل أساســى مثل الأســلوب المقدم من شــركة Agile (Martin, 2003). يعتمد هذا الأسلوب على فكرة تجهيز برنامج الاختبار أولاً قبل البدء فى برمجة التطبيق. وعلى هذا فإنه يتم تطوير برامج التطبيق بغرض اجتياز الاختبارات المعدة مســبقاً ممــا يؤدى إلى ضمان كفاءة برامــج التطبيق وتأديتها للوظائف المطلوبــة بنجاح. تعتبر هذه العملية تكرارية، حيث يتــم إعداد برنامج اختبار أولــى لبعض الوظائف الأساســية بالنظام وتجهيز حالات الاختبار لها. وبعد ذلك يتم كتابة أكواد البرنامج المنفذ لتلك الوظائف وتنفيذه لاجتياز الاختبار المعد له مسبقاً. وعندما يجتاز البرنامج الاختبار يتم توسعة برنامج الاختبار الأولى بإضافــة حالات اختبار أخرى لوظائف جديدة فى النظام، ويتم أيضاً توســعة البرنامــج التطبيقى المقابل له، ويتم تنفيــذه لاجتياز الاختبارات المعدة له وهكذا حتى

يتـــم الانتهاء من تطبيق جميع وظائف النظام. والســؤال الآن هو أين يتم وضع الكود الخــاص بحالات الاختبار بالنســبة للكود الخاص بالتطبيق المطلــوب اختباره؟ هناك مجموعة من الأســاليب المتبعة لحل هذا السؤال. أحد هذه الأساليب هو وضع الكود الخاص بحالة الاختبار Test Case داخل الطريقة الرئيســية () main لكل صنف يتم اختباره. الأسلوب الثانى هو وضع حالة الاختبار فى صنف داخلى inner Class داخل الصنف المطلوب اختباره. والســبب فى ذلك هــو أن الكائن الداخلى (كائن الاختبار) يمكنــه أن يتعامل بحرية مع كل عناصر الكائن الخارجــى له (الكائن تحت الاختبار). يوضح شكل (٩-١٠) إطاراً عاماً لنموذج الاختبار. يحتوى النموذج على صنفى اختبار أساســين هما TestSuit ،Test Case وكلاهما يكون امتداداً أو تطبيقاً لصنف واجهة الاختبار test الذى يصف الطريقة () run لتنفيذ الاختبار. يعتبر الصنف TestSuit مــن الأصناف التى تحتوى علــى مكونــات Composite Class وهو يمكن أن يتضمن أكثــر من حالة اختبــار، أى أن الكائــن TestSuit يمكن أن يحتوى علــى عدة كائنات اختبار TestCase. أى أن هذا النموذج يسمح بحالات اختبار فردية من خلال الكائن TestCase ويسمح أيضاً بحالات اختبار مركبة من خلال كائنات TestSuit.



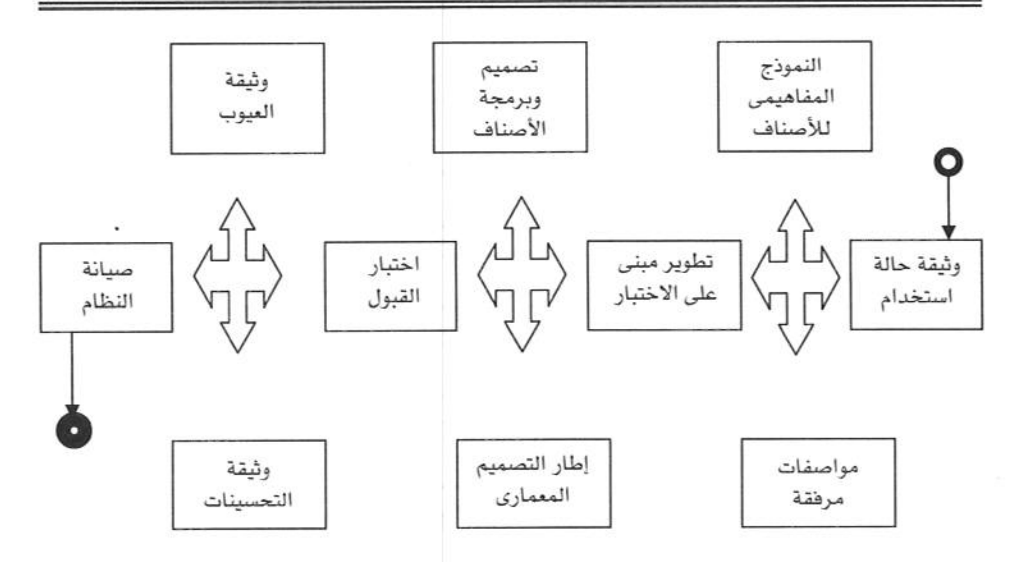
شكل (٩-١٠) إطار عام لنموذج الاختبار

يتم تطبيق وحدة الاختبار مــن خلال صنف فرعى هو امتداد للصنف TestCase. يأخــذ هذا الصنف الفرعى الاســم xxxTest كما هو موضح بالشــكل ويقوم بإضافة

طــرق جديدة لاختبار الوحدة وتكــون امتداداً للطرق الموجــودة بالصنف TestCase. تقوم الطريقة () Setup للكائن TestCase بإنشــاء كائنــات وحدة الاختبار التى تمثل الهــدف لكود الاختبار. وتقوم الطريقــة () TearDown بإزالة كائنات وحدة الاختبار مــن الذاكــرة بعد انتهاء كل حالة اختبار. وتقوم الطريقــة () RunTest بتنفيذ وحدة الاختبار وذلك من خلال استدعاء حالة الاختبار المطلوبة داخل الكائن xxxTest.

علــى الرغم من أن عملية التطوير المبنى على الاختبــار تركز على اختبار وحدات صغيــرة مثــل الأصناف (كل صنف على حدة أو مجوعة بســيطة من الأصناف) فإنها لا تستطيع أن تكتشف الأخطاء فى تطبيق المتطلبات الوظيفية للنظام العام أو تكتشف عــدم إنجاز بعض المهام الوظيفية. وعلى هذا فــإن اختبار الوحدات الصغيرة Units Test يسمى «اختبار الصندوق الأبيض White Box Test» لأنه يركز على كيفية تطبيق المهــام أو الوظائف داخل كل صنف أو وحدة. أما النوع الآخر من الاختبار الذى يطلق عليه اســم «اختبار الصندوق الأســود Black Box Test» فهــو يهتم بعملية التأكد من تحقيــق المتطلبات الوظيفية للنظام بغض النظر عن كيفيــة تحقيقها ولكنه يركز على اكتمال المتطلبات ومطابقتها لمواصفات متطلبات النظام. وعلى هذا فإن هذا النوع من الاختبار يتم بواســطة العميل Customer والمستخدم User للنظام ولذلك فهو يسمى «اختبار القبول Acceptance Test» من ناحية المســتفيد. يوضح شــكل (٩-١١) دور عمليــات الاختبار فى دورة حياة تطوير التطبيقــات. تبدأ عملية التطوير المبنية على الاختبــار بمجموعة من وثائق حالات الاســتخدام Use Case Documents والنموذج المفاهيمى للأصناف Conceptual Class Model وبعض المواصفات الإضافية. وبعد ذلك يتم تجهيز حالات الاختبار Test Cases التى ســيتم اســتخدامها فى بناء برامج وحــدات الاختبار Test Units. وبعــد انتهاء عمليات تطوير واختبار وحدات التطبيق يتم تنفيذ اختبار القبول على التطبيق النهائى.

ولأن اختبار القبول لا يهتم بكيفية التطبيق فإنه يتم تجهيزه بشكل منفصل عن عملية التطبيق وذلك من خلال مجموعة المســتخدمين والمســتفيدين من النظام. يتم تجهيز مواصفات الاختبار فى شــكل مجموعة من سيناريوهات الاختبار Test Scripts. يتم تسجيل مجموعات العيوب الناتجة عن اختبار القبول فى شكل وثيقة العيوب Defects Document ليتم تصحيحها من قبل المطورين للتطبيق. ويتم تسجيل مجموعة الملاحظات حول التحسينات المطلوبة فى شكل وثيقة التحسينات Enhancements Document.



شكل (٩-١١) دور الاختبار فى تطوير التطبيقات

ويتم تطبيق تلك التحسينات وتصحيح العيوب من خلال عمليات الصيانة للتطبيق. يتضمن اختبار القبول مجموعة من نقاط التحقق Verification Points التى تستخدم فى تحديد النتائج المتوقعة من ناحية المستفيد بالنسبة لسلوك النظام الوظيفى مقابل مجموعة المدخلات والشروط وقواعد العمل المتضمنة فى الوظيفة تحت الاختبار. يتضمن اختبار القبول مواصفات البيانات التى سيتم استخدامها فى عملية الاختبار والنتائج المتوقعة المقابلة لتلك البيانات. يوضح شكل (٩-١٢) محتوى وثيقة سيناريو اختبار القبول لتطبيق إدارة البريد الإلكترونى EM. تحتوى الوثيقة على ثلاثة أعمدة: الأول يصف رقم الخطوة Step (S) أو نقطة التحقق Verification Point (V). والعمود الثانى يصف النتيجة المتوقعة من التحقق True، False، Unknown. من الممكن أن تكون نتيجة التحقق فى الحالات الأولية «غير معروفة Unknown» وهنا يتم التدخل بالتحقق اليدوى من جانب المستفيد أو المطور لاستكمال الاختبار وتحديد النتيجة بدقة True/False. يحتوى العمود الثالث فى الوثيقة على وصف لخطوة الاختبار أو نقطة التحقق وتأخذ نقطة التحقق صيغة السؤال دائماً. تهتم وثيقة الاختبار الحالية بما يلى:

- اختبار عملية الدخول على النظام بنجاح (4V، 6V) أو الفشل فيها (2V).
- عرض الرسائل المرسلة للموظف المتصل (8V، 10V، 12V).

- إرسال رسالة معينة (14V).
- عرض آخر رسالة تم إرسالها (19V).
- إعادة قاعدة بيانات الرسائل إلى الوضع السابق (16V).
- التحقق من الخروج من النظام بنجاح (21V).

الوصف	النتيجة	الخطوة/نقطة التحقق
أدخل اسم المستخدم وكلمة المرور "login" غير صحيحة هل نجحت عملية الدخول ؟	False	1S 2V
أدخل اسم المستخدم وكلمة المرور "login" صحيحة هل نجحت عملية الدخول ؟	True	3S 4V
تحقق من وجود الموظف "get employee" هل الموظف موجود؟	True	5S 6V
استعرض عدد الرسائل المرسلة "count of messages" هل تم عرض العدد؟	True	7S 8V
استعرض قائمة الرسائل المرسلة "list of messages" هل تم عرض القائمة؟	True	9S 10V
استعرض بيانات رسالة مرسلة "get outmessage" هل تم عرض بيانات الرسالة؟	True	11S 12V
أرسل الرسالة السابقة لنفسك "email outmessage" هل تم وصول الرسالة على عنوانك؟	Unknown	13S 14V
استعرض بيانات الرسالة المرسلة فى الخطوة السابقة "get outmessage" هل تم عرض بيانات الرسالة؟	False	15S 16V
استرجع قاعدة البيانات للوضع السابق لها قبل إجراء الاختبار استعرض بيانات الرسالة المرسلة فى الخطوة رقم 14 "get outmessage" هل تم عرض بيانات الرسالة؟	True	17S 18S 19V
أغلق التطبيق EM هل تم الخروج بنجاح؟	True	20S 21V
اخرج أو أعد تنفيذ الاختبار		22S

شكل (٩-١٢) وثيقة سيناريو اختبار القبول للتطبيق EM

٩-٣ اختبار البرمجيات Software Testing:

تبـدأ عملية الاختبار للبرامج بعد بداية عملية البرمجة بفترة قصيرة وتسـتمر معها حتـى نهاية اختبـار جميع البرامج المنتجة. فبعد الانتهاء مـن برمجة أى وحدة برمجية Module يتـم اختبارها بمفردها أولاً، ثم اختبارها باعتبارها جزءاً من مهمة أو برنامجاً أكبر، ثم أخيراً اختبارها على أنها جزء من النظام العام. ومع أن عملية الاختبار تتم فى أثنـاء عملية التنفيذ، ولكن فى الحقيقة يتم التخطيط والتجهيز لها فى مراحل سـابقة، وذلـك بتحديد الوظائف المطلوب اختبارها وإعداد بيانات الاختبار لها. تبدأ أولاً عملية التخطيط لاختبار البرامج فى مرحلة التحليل باختيار الوظائف وإعداد بيانات الاختبار، ثم فى مرحلة التصميم يتم إعداد خطة الاختبار للوحدات البرمجية وخطة اختبار التكامل بـين الوحدات، ثم خطة الاختبـار العامة للنظام. وفى مرحلة التنفيـذ يتم تطبيق تلك الخطط فعليـاً لاختبار النظام. يعتبر الغرض الحقيقى من تلك الخطة الثلاثية لاختبار النظام هو تحسين عملية الاتصال بين الأفراد القائمين على تطوير المشروع وتحديد دور كل فـرد منهـم فى عملية الاختبار. وعلى هذا فإنه يمكننا اعتبار أن عملية الاختبار هى بمنزلة مشروع داخل المشروع الحالى؛ لأنها تحتاج إلى خطة ومراحل لتنفيذها ومتابعتها. يمكننا تصنيف أنواع الاختبارات المختلفة فيما يلى: (Frank Tsui, 2004).

نوع الاختبار	يدوى	آلى
ساكن Static	فحص Inspection	تدقيق لغوى Syntax Checking
متحرك Dynamic	مراجعات تتابعية walkthroughs تدقيق مكتبى Desk checking	اختبار الوحدة Unit Test اختبار التكامل Integration Test اختبار النظام System Test

يمكن أن تكون عملية الاختبار ساكنة Static أى تكون قبل تنفيذ البرامج، أو تكون Dynamic أى أنها تكون وقت التنفيذ للبرامج. ويمكن كذلك تصنيف عملية الاختبار فهى يمكن أن تتم يدوياً Manual أو آلياً باسـتخدام الحاسب الآلى. فيما يلى عرض مختصر لأنواع الاختبار.

الفحص Inspection:

هو أسلوب اختبار يدوى ساكن Static يقوم فيه الشخص المختص بفحص البرنامج واصطياد الأخطاء اللغوية المعروفة Syntax errors. تؤدى هذه الطريقة إلى اكتشاف نحو (٦٠٪) أو أكثر من الأخطاء الموجودة فى البرنامج.

المراجعة التتابعية Walkthrough:

هى أسـلوب اختبار يدوى سـاكن يتم فيه توثيق ومراجعة أجـزاء برمجية صغيرة بالاشـتراك مع المبرمج المنتج لها وذلك بطريقة دورية لتجنب أغلب الأخطاء أولاً بأول وقبـل عمليـة اختبار البرنامج بأكملـه. بالطبع تؤدى هذه الطريقـة إلى تقليل كمية الأخطاء المتوقعة فى أثناء اختبار البرامج الكبيرة الحجم.

تدقيق مكتبى Desk checking:

هى طريقة يدوية يتم فيها مراجعة البرنامج مكتبياً باستخدام الورقة والقلم، حيث يقـوم المراجـع بتنفيذ كل خطوة من البرنامج يدوياً باسـتخدام حالات اختبار محددة Test Cases. وهنـا يقوم المراجع بدور الحاسـب، فيقوم بتنفيـذ كل خطوة ومقارنة النتائج.

تدقيق لغوى Syntax Checking:

هو أسلوب اختبار آلى ولكن ساكن، حيث يتم تدقيق البرنامج لغوياً فقط باستخدام مترجمات لغات البرمجة Compilers. ويتم ذلك قبل تنفيذ البرنامج للتأكد من خلوه من الأخطاء اللغوية ولكن بالقطع لا يمكن اكتشاف الأخطاء المنطقية بالبرنامج.

اختبار الوحدة Unit Test:

هو أسـلوب اختبار آلـى يتم فيه تنفيـذ الوحدات البرمجيـة الفرعية Modules منفصلة. ويتم اكتشاف الأخطاء الموجودة بكل وحدة برمجية على حدة.

اختبار التكامل Integration Test:

هو أسلوب اختبار آلى يتم فيه تنفيذ واختبار تكامل الوحدات البرمجية معاً بالتدريج بأسـلوب Top-Down. فنبدأ بتنفيذ واختبار الوحدة البرمجية الأساسية Root مع إحدى الوحدات البرمجية الفرعية. وبعد ذلك يتم إضافة وحدة برمجية فرعية أخرى واختبارها مع الوحدات السـابقة، وهكذا حتى يتم الانتهاء من اختبار جميع الوحدات البـرمجـية معاً.

اختبار النظام System Test:

هو أسـلوب اختبار آلى مشابه تماماً للأسلوب السابق ولكن مع الفارق أنه فى هذه الحالة يتم تجميع برامج Programs تؤدى وظيفة متكاملة بعضها مع بعض لاختبارها، بـدلاً من تجميع وحدات فرعية صغيـرة Modules بعضها مع بعض كما فى الطريقة السابقة. ويتم اختبار تكامل البرامج بعضها مع بعض أيضاً بالتدريج، كما سبق، حتى يتم اختبار النظام العام كله System.

اختبار القبول Acceptance Test:

بعـد الانتهاء من عملية اختبار البرامج يأتى دور اختبار القبول Acceptance Test وهـو يتم عادة من قبل المسـتخدم User. وهذا الاختبار للنظـام يأتى على مرحلتين أساسيتين هما:

١- اختبار أولى Alpha Test:

يتم فيه اختبار النظام باستخدام بيانات مصممة تحاكى بيانات النظام Simulated data. يتـم إجـراء عدة اختبـارات للنظام فى هذه المرحلة للتأكـد من تحقق معايير الصحة والأمن الأساسية لأى نظام مثل:

اختبار الاسترجاع Recovery Test:

يتـم فيه إجبار النظام على الانهيار للتحقق من إمكانية اسـترجاع النظام لبياناته بأمان وبدون فقد للمعلومات.

اختبار الأمن Security Test:

يتم فيه اختبار آلية حماية النظام من الاختراق.

اختبار التحميل Stress Test:

يتم فيه تحميل النظام بعدد كبير من العمليات من عدة مستخدمين فى وقت واحد أو إدخـال بيانات غير مكتملة، ثم تحليل النتائج لقيـاس مدى تحمل النظام للتحميل الزائد عند التشغيل.

اختبار الأداء Performance Test:

يتم فيه تحديد كيفية أداء النظام فى بيئات عمل مختلفة. فمثلاً يتم اختبار النظام من خلال أنظمة تشغيل مختلفة ومواصفات شبكات مختلفة وأجهزة مختلفة.

٢- اختبار تمهيدى Beta Test:

يتـم فى هـذه المرحلة تشـغيل النظام عند المسـتخدم فى بيئة العمـل الطبيعية، باسـتخدام بيانات طبيعية لاختبار مـدى ملاءمة البرامج والتوثيـق والتدريب المقدم لمتطلبات المسـتخدم. ويتم أيضاً فى هذه المرحلة اكتشاف الأخطاء التى لم تظهر فى مراحل الاختبار السـابقة لكونها تتعامل مع بيانات طبيعية لم تكن فى الحسـبان عند تجهيز بيانات الاختبار فى المراحل السابقة.

الفصل العاشر
تصميم واجهات المستخدم
User Interface Design

بينمـــا يهتم التصميم المعمارى للنظـــام بالهيكل الداخلى والأعمال التفصيلية داخل نظام الأعمال، فإن تصميم واجهات المستخدم يهتم بالمظهر الخارجى للنظام وهو يؤدى إلى جذب العميل وكســب رضاه. فبالإضافــة إلى المجهود المبذول فى تحقيق أهداف نظـــام الأعمال بدقة وكفاءة، فإنه لابد من بذل مجهود آخر لا يقل عنه أهمية لتصميم واجهة استخدام النظام UI تتســم بالسهولة والجاذبية وتساعد المستخدم على تنفيذ مهامه بشــكل أسرع وأبسط من الطرق التقليدية التى كان يستخدمها من قبل. تتسم معظـــم تطبيقات الأعمال باعتمادها أســلوب الخادم والعميـــل Client/Server الذى يتكـــون من عـــدة طبقات مثل: طبقة خادم التطبيقـــات Application Server وطبقة خادم الويـــب Web Server، وطبقة خادم قواعد البيانات Database Server وطبقة العميل Client. يمكننا تصنيف طبقة العميل إلى صنفين:

عميل مبرمج Programmable Client:

هـــو العميل الذى يمتلك نســـخة من التطبيق والبرامج المنفـــذة له ويمكن أيضاً أن يمتلك قواعد البيانات التى يتطلبها التطبيق، أى أنه يتم نشر جميع محتويات التطبيق محلياً عند العميل. وأحياناً لا يتم نشـــر قواعد البيانات عند العميل ولكن يتم تحميل البيانـــات من خادم قواعد البيانات المتصل به إلى ذاكرة التطبيق ويتم معالجتها محلياً ثم إعادة تسجيلها فى قواعد البيانات مرة أخرى عند الحاجة لذلك.

عميل متصفح Browser Client:

هـــو عميل يعتمـــد دائماً على خـــادم الويب فى تحميل البيانـــات المطلوب عرضها علـــى صفحة العميل، وكذلك إرســـال البيانات المدخلة من خـــلال صفحة العميل إلى الخادم لمعالجتها وتخزين البيانات الضرورية فى قاعدة البيانات واسترجاع النتائج إلى العميل عند الضرورة. أى أن العميل المتصفح لا يقوم بأى عمليات معالجة باســـتثناء بعض عمليات التدقيق البسيطة التى يجريها على البيانات المدخلة للتأكد من اكتمالها ومطابقتهـــا للمواصفات المطلوبة قبل إرســـالها للمعالجة عند الخادم. تأخذ البيانات

المقدمة عند العميل شـــكل صفحات HTML، وســـوف نقدم الآن عرضاً لبعض طرق تصميم وبرمجة واجهات المستخدم باستخدام لغة جافا.

١٠-١ إرشادات لتصميم واجهة المستخدم User Interface Design Guidelines:

تركـــز عمليـــة تصميم واجهات المســـتخدم للنظـــام على صفة "اســـتخدام النظام Usability" التـــى تمثل الشـــكل الخارجى للنظام. يتســـم التصميـــم الجيد لواجهات المستخدم بالخصائص التالية:

- سهولة الاستخدام.
- سهولة التعلم.
- زيادة كفاءة المستخدم.
- سرعة إنجاز المهام.
- تقليل الأخطاء فى أثناء الاستخدام.

هناك مجموعة من الإرشـــادات التى تساعد على تصميم واجهات المستخدم بشكل جيـــد يحقق الهـــدف منها. يمكننا تقديم هذه الإرشـــادات من خـــلال النقاط الأربع التالية:

١- تحكم المستخدم:

من الإرشادات المهمة جداً فى أثناء تصميم واجهات المستخدم هو إعطاء المستخدم ميـــزات التحكم فى تشـــغيل النظـــام وتوجيهه من خـــلال مجموعـــة التفاعلات بين المســـتخدم والنظام. وبالفعل نجد أن واجهات المستخدم الحديثة خصوصاً التى تعمل فى بيئة الإنترنت Web Based توفر للمســـتخدم ميزات التحكم فى البرنامج وتحديد الوجهـــة التى يرغب فيهـــا لتنفيذ المهام وليس العكس كمـــا كان الحال فى التطبيقات القديمة. أما الآن فباستخدام التقنيات الحديثة أصبح التحكم فى يد المستخدم عند تشـــغيل التطبيق وأصبح هو الذى يحدد للبرنامج مـــا هو المطلوب تنفيذه فى الخطوة التاليـــة، ويمكن أيضاً أن يوقف عملية التنفيذ قبل انتهائها. ومن المهم أن تكون واجهة المستخدم سهلة الاســـتخدام وتستخدم مصطلحات وتعبيرات نظام الأعمال الحقيقية وليســـت مصطلحات الحاسب الآلى. ويجب أن يراعى مصمم واجهة المستخدم عملية

إعلام أو تنبيه المستخدم بوضوح عن بعض التغييرات التى يمكن أن تحدث فى بعض مكونات واجهة المستخدم مثل: (عناصر القوائم Menu Items أو مفاتيح التحكم Command Buttons) نتيجة لحالة التطبيق الحالية فى أثناء تنفيذ المهام. ومن الضرورى أيضاً أن يراعى عند تصميم واجهة المستخدم إظهار معلومات للمستخدم عند قيام البرنامج بتنفيذ بعض المهام التى لا يجب توقيفها قبل استكمالها لضمان سلامة البيانات ومن ثم كفاءة النظام.

٢- اتساق الواجهات:

يجب أن تتوافق واجهات المستخدم مع المعايير المستخدمة داخل المؤسسة من ناحية الأسماء المستخدمة لمكونات الواجهة والاختصارات المسموحة والأماكن المناسبة لوضع المحتويات فى الواجهة، وكذلك تحديد عناصر التحكم المناسبة مثل: قوائم التحكم menus ومفاتيح التحكم buttons وبعض مفاتيح الوظائف فى لوحة الإدخال keyboard functions. يجب التأكد من تحقق عملية التوافق هذه فى جميع واجهات التطبيق الخاصة بالمؤسسة مما يضمن اتساق واجهات النظام العام. يجب على مصمم واجهات التطبيق تجنب استخدام اتجاهات غريبة وغير مألوفة للمستخدم بغرض الإبداع أو الجاذبية من وجهة نظره، ولكنها يمكن أن تعوق عملية استخدام التطبيق مما يؤثر سلباً فى كفاءة استخدام النظام. ومع أن إرشادات تصميم الواجهات تدعم فكرة استخدام الأساليب التى اعتاد عليها المستخدم عند التشغيل، ولكن من الأفضل الاستفادة من الأساليب الحديثة والتسهيلات التى تقدمها التقنيات الحديثة التى تعزز من جودة وكفاءة استخدام النظام. تتميز أيضاً الواجهات جيدة التصميم باتساق الألوان وحسن استخدامها واستخدام المسافات المناسبة بين عناصر الواجهة وتجميع العناصر ذات العلاقة فى مكان واحد أو بمعنى آخر تصنيف محتويات الواجهة فى شكل مجموعات مما يسهل من عملية استخدامها.

٣- واجهات متسامحة:

يجب أن تتسم واجهات التطبيق بالتسامح نحو تصرفات المستخدم غير الطبيعية مثل: إدخال بيانات غير صحيحة أو محاولة تنفيذ مهام غير مناسبة للحالة الحالية للتطبيق. وبمعنى آخر فإن الواجهة المتسامحة هى التى تشجع المستخدم على استكشاف الخيارات المختلفة للتطبيق من دون التأثير فى استمرار تشغيل النظام

بنجاح. يتم تنفيذ ذلك عن طريق استخدام رسائل توجيهية للمستخدم عند حدوث أحد الاستثناءات Exceptions فى أثناء التشغيل. تختلف طبيعة رد فعل التطبيق نحو الأفعال غير المتوقعة من ناحية المستخدم. فأحياناً يكون رد الفعل فى شكل رسالة تنبيهية، وأحياناً يكون من خلال عدم السماح أساساً بحدوث هذا الفعل بعدم تمكين المستخدم من إجرائه. وفى جميع الأحوال يجب أن يتميز التطبيق بالصمود أمام هذه الأفعال غير المتوقعة أى أنها لا يجب أن تؤدى إلى سقوط النظام. ولكن يجب أن يتم إعلام المستخدم بطبيعة المشكلة الحالية نتيجة استخدامه للتطبيق ولماذا لا يستطيع التطبيق تنفيذ الأفعال المطلوبة، ويفضل أيضاً أن يخبره بكيفية تصحيح الفعل أو عرض الأفعال البديلة والمسموح بها فى التطبيق.

٤- تكيف الواجهة:

من مواصفات التصميم الجيد لواجهة المستخدم أن تكون متوافقة مع أكبر عدد من المستخدمين. فمن الممكن أن يكون هناك اختلاف فى إمكانيات المستخدمين للنظام من حيث الخبرة فى استخدام الواجهات والتفاعل معها أو من ناحية الوظائف التى يمكنه أن يستخدمها أو المقدرة الشخصية للأشخاص الذين لديهم إعاقة أو قدرات خاصة. وهنا يجب أن يتم الأخذ فى الاعتبار عند تصميم واجهات النظام أن تكون متكيفة مع إمكانيات المستخدم، أى أنها تسمح للمستخدم بتعديل مواصفات الواجهة حسب قدراته وصلاحياته فى استخدام النظام. ومن الممكن أيضاً أن يتم تنفيذ النظام من خلال أنظمة تشغيل مختلفة، وهنا يجب أن يتكيف التطبيق مع نظام التشغيل الحالى. ويتضمن التكيف أيضاً تغيير لغة التطبيق لتلائم لغة المستخدم أو تغيير الخطوط وحجمها أو تغيير عملة المستخدم أو التغيير فى الوقت المحلى وشكل التاريخ أو شكل أرقام الهواتف وغيرها من الخصائص التى يمكن تغييرها فى واجهات التطبيق لتلائم طبيعة المستخدم. ومن الأشياء المهمة أيضاً فى حالة المستخدمين ذوى القدرات الخاصة أن تتكيف واجهة المستخدم لتلائم قدراتهم. ففى حالة المستخدم الضرير يجب أن تتضمن الواجهة إمكانيات التعامل مع الأصوات أو استخدام طريقة برايل. وفى حالة المستخدم الأصم يجب أن تعتمد الواجهة على الإمكانيات المرئية بشكل موسع وتعويض أى معلومات صوتية.

١٠-٢ مكونات واجهة المستخدم User Interface Components:

تعتمـد عملية برمجة واجهات التطبيق على الاسـتفادة من مكتبة الأصناف Class Library الجاهـزة والمتوافرة بشـكل كبيـر من خلال عدد من شـركات البرمجيات. تحتـوى مكتبة الأصناف على عدد كبير من المكونات التى تسـتخدم فى بناء واجهات المسـتخدم بأنواعها المختلفة. فمثلاً تحتوى مكتبة الأصناف فى لغة جافا على حزمة خاصة بمكونات واجهات المسـتخدم تسـمى "Swing" وحزمة أخرى تسمى "AWT". تمكننا أداة Swing من تطوير تطبيقات تتسم بإمكانية تشغيلها على عدة أنظمة تشغيل وإمكانية تخصيص واجهات المستخدم حسب طبيعة التطبيق والمستخدم. تحتوى أداة Swing علـى مجموعة من المكونـات التى تبدأ دائماً بالحـرف J مثل: JMenuBar, JButton, JDialog، وغيرها التى يتم اسـتخدامها فى أثناء تصميم وجهات التطبيق. يمكننا تصنيف هذه المكونات فى شكل مجموعات على النحو التالى:

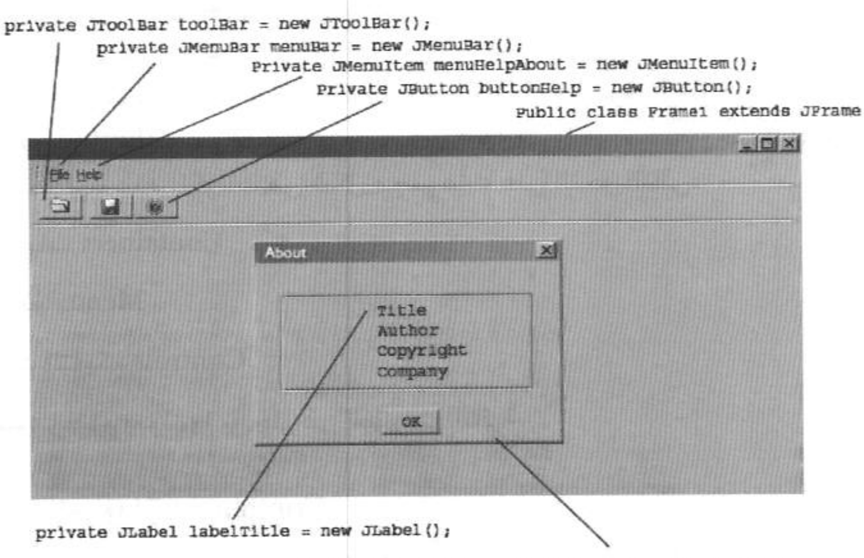
- الحاويات Containers.
- القوائم Menus.
- أدوات التحكم Controls.

وسوف نقوم بعرضها بإيجاز فى الفقرات التالية.

١٠-٢-١ الحاويات Containers:

تمثل الحاويات مجموعة الكائنات التى توضع على واجهة سـطح المكتب ويمكن أن تحتـوى على مكونات أخرى مثل القوائـم وأدوات التحكم وحتى يمكن أن تحتوى على حاويـات أخرى. تمثل المكونـات Windows، Dialogs، Panes، Panels بعض الأمثلة على الحاويات المسـتخدمة فى تصميم واجهات المسـتخدم بلغة جافا. تحتوى حزمة Swing على مجموعة من الأصناف التى تستخدم لإنتاج كائنات الحاويات. بعض هذه الكائنات يستخدم فى المسـتوى الأول للواجهة مثل: JWindow, JFrame, JDialog، JApplet، والبعـض الآخر يسـتخدم فى المسـتويات التالية للواجهـة مثل: JTable, JPanel, JOptionPane, JScrollPane وغيرهـا. تحتـوى الواجهة عادة على نافذة رئيسية واحدة ومجموعة من النوافذ الفرعية. ويستخدم المكون JPanel لتطبيق إحدى النوافذ الفرعية التى تقوم بعرض رسائل للمستخدم أو لإدخال بيانات مثل Dialog Box. يوضح شـكل (١٠-١) مثالاً على اسـتخدام نافذة فرعية داخل نافذة رئيسية. تعتبر

النافذة الرئيســية من المكون JFrame والنافذة الفرعية مــن المكون JPanel. تحتوى النافذة الرئيســية على مكون القائمة JMenuBar ومكون شــريط الأدوات JToolBar وبعض مفاتيــح التحكم فى النافذة مثل: التصغير والتكبير والإغلاق. ويحتوى مكون القائمــة على مكونات أخرى فرعية مثل JMenuItem، وكذلك يحتوى مكون شــريط الأدوات على مجموعة من مفاتيح التحكم JButtons. تتميز النافذة الرئيسية بإمكانية تغيير حجمها من أى جنب أو من الزوايا ويمكن أيضاً تحريكها من شريط العنوان.



شكل (١٠-١) استخدام النوافذ فى واجهات المستخدم

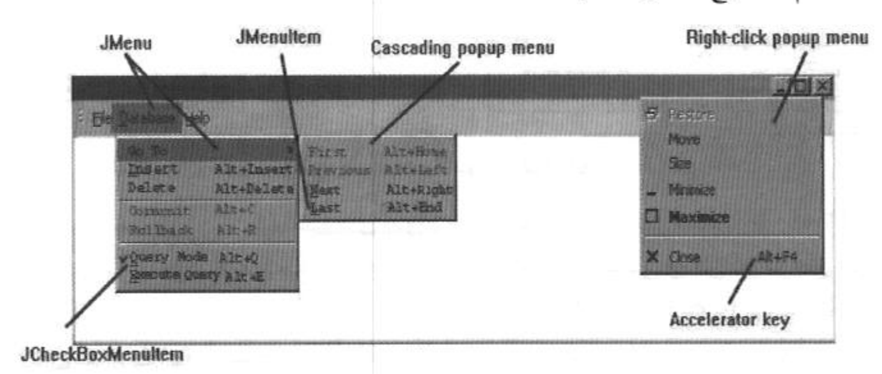
. يوجد مكون آخر يســمى JApplet يســتخدم فى تنفيذ برامج داخل المتصفح، ويتم تحميل تلك البرامج من خادم الويب إلى صفحة المتصفح . تقدم أداة Swing أسلوباً لتنظيــم وضع المكونــات داخل الواجهــة يســمى "LayoutManager" الذى يتضمن مكونــات مثــل BorderLayout، FlowLayout، GridLayout، BoxLayout. يعتبــر أسلوب BorderLayout هو الأسلوب الافتراضى لتنظيم وضع المكونات داخل حاويات واجهة التطبيق. تقدم أداة Swing أيضاً صنفاً خاصاً يدعم وضع النوافذ فى طبقات Layers ويســمى JLayerPane. يحتــوى الصنف على طــرق خاصة بعمليات تحريك المكونات من الأمام إلى الخلف والعكس أو وضعها فى أماكن معينة. يتم وضع المكونات

فى الطبقة الافتراضية Default Layer بشكل متداخل Overlap ويتم تحديد النافذة الأمامية للواجهة حسب اختيار المستخدم. ويسمح الصنف JLayerPane بتعريف طبقات بمواصفات خاصة مثل:

- طبقة عائمة Palette Layer: تظهر فوق الطبقة الافتراضية.
- طبقة إجبارية Modal Layer: تظهر على رأس كل النوافذ النشطة مثل شريط الأدوات.
- طبقة منبثقة Popup Layer: تظهر مؤقتاً فوق طبقات أخرى.
- طبقة سحب Drag Layer: تظهر المكون المطلوب سحبه قبل إلقائه فى طبقة أخرى.

١٠-٢-٢ القوائم Menus:

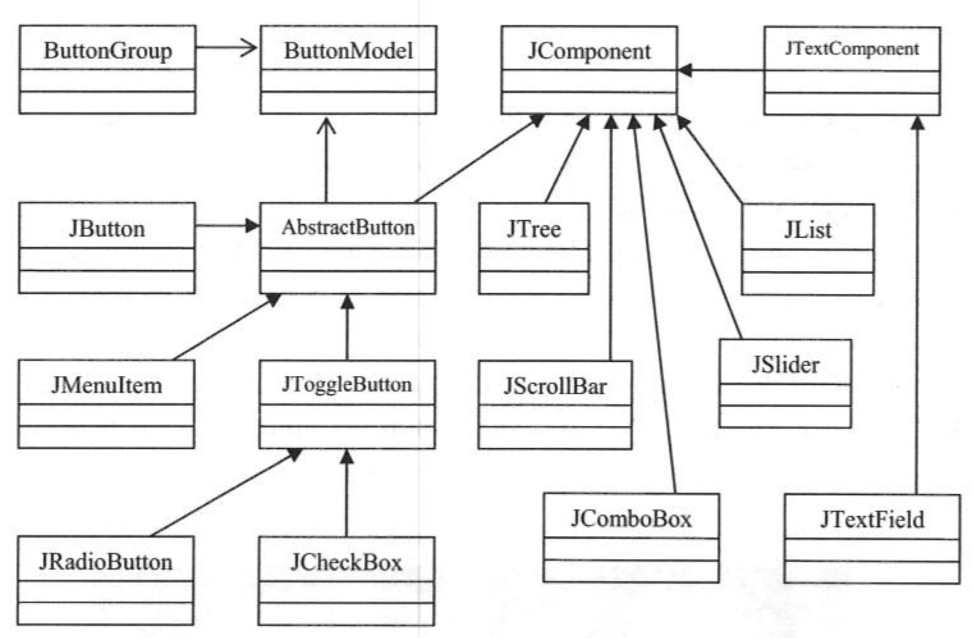
تحتوى معظم نوافذ واجهات التطبيق على مكون القائمة الذى يتضمن مجموعة من عناصر التحكم فى تشغيل التطبيق. تقدم أداة Swing مجموعة من الأصناف الخاصة بالقوائم. يوضح شكل (١٠-٢) مثالاً لواجهة تطبيق تحتوى على شريط القائمة JMenuBar الذى يتضمن كائنات JMenu. ويحتوى كل كائن JMenu على مجموعة من عناصر القائمة JMenuItem مثل Insert أو GoTo الذى يحتوى على عناصر أخرى فرعية فى شكل قائمة منبثقة JPopUpMenu. يمكن أن يأخذ عنصر القائمة JMenuItem شكل صندوق الاختيار JCheckBoxMenuItem أو شكل مفتاح الاختيار JRadioButtonMenuItem. ويمكن أيضاً تشغيل عناصر القائمة باستخدام مفاتيح السرعة Accelerator Keys.



شكل (١٠-٢) استخدام القوائم فى واجهة التطبيق

١٠-٢-٣ أدوات التحكم Controls:

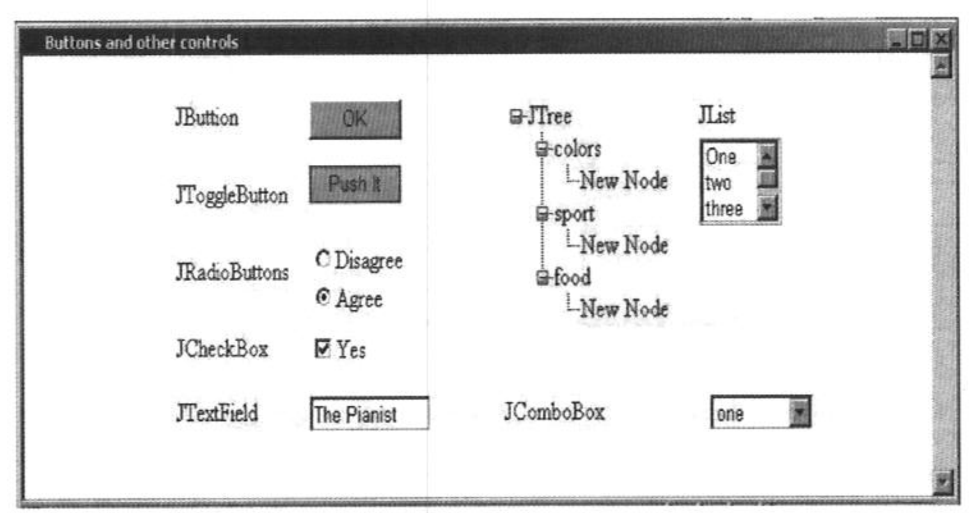
تعتبــر أدوات التحكــم بمنزلة القلب لنموذج أداة Swing حيــث إنها تقوم بعمليات التقاط البيانات والأفعال التى يطلبها المستخدم وتفسيرها وتطبيقها من خلال واجهة التطبيق. يوضح شــكل (١٠-٣) نمــوذج الأصناف لأدوات التحكــم فى أداة Swing. يوضح الشــكل وجود أربعة أنواع من مفاتيح التحكم هى: JButton, JRadioButton, JCheckBox, JMenuItem، وهــى تــرث مــن الصنف المجــرد AbstractButton. بالإضافــة إلــى مجموعة أخــرى منوعة مــن أدوات التحكــم مثــل: JTree, JList, JComboBox, JSlider, JScrollBar وهى ترث من الصنف المجرد JComponent.



شكل (١٠-٣) نموذج الأصناف لأدوات التحكم فى أداة Swing

ويوضــح شــكل (١٠-٤) أشــكال أدوات التحكم المذكورة فى شــكل (١٠-٣). يتم تطبيــق أدوات التحكــم من خلال مجموعة من الأحداث التــى يمكن أن ترافق الأداة. فمثــلاً يتضمن كائن التحكم JButton الحــدث () Click والحدث () DoubleClick وغيره من الأحداث. تســتخدم الأداة JcheckBox للاختيار المتعدد، فى حين تستخدم الأداة JRadioButton لاختيار واحد فقط من مجموعة الاختيارات. تســتخدم الأداة JComboBox لاســتعراض قائمة منســدلة من العناصر واختيار أحدها فقط ســواء

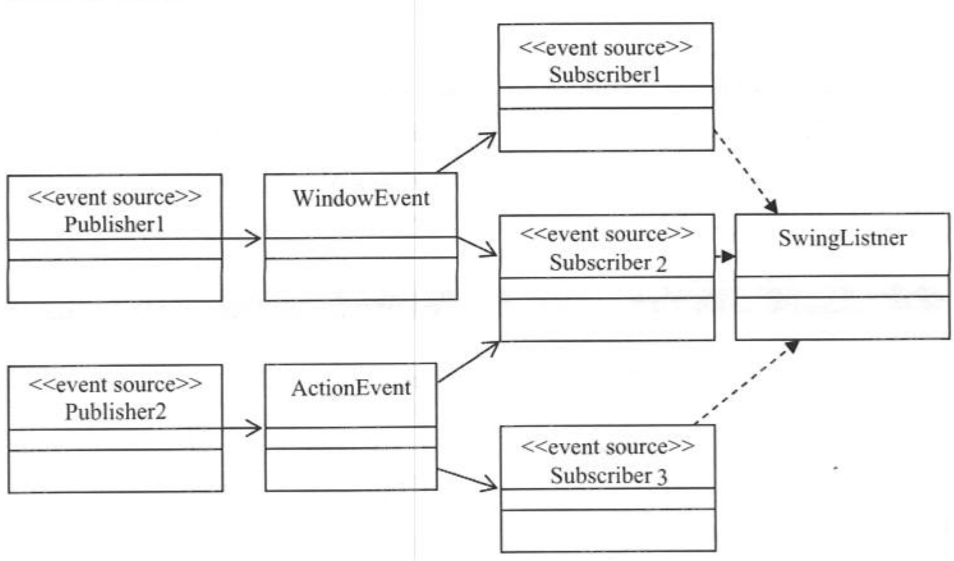
بالنقر على العنصر نفسه أو بكتابة جزء منه فى صندوق النص الخاص بالأداة. أما الأداة JList فتستخدم لعرض قائمة عناصر بحجم ثابت Fixed Size ويمكن اختيار مجموعة من العناصر فى الوقت نفسه. تستخدم أداة JSlider للتحكم فى بعض المدخلات مثل التحكم فى الصوت Volume Control. وتستخدم أداة JScrollBar لإمكانية تصفح باقى محتويات أى مكون على الواجهة له حجم أقل من حجم المحتويات له.



شكل (١٠-٤) أدوات التحكم فى أداة Swing

١٠-٣ معالجة أحداث واجهات المستخدم User Interface Event Handling:

تحتاج جميع الأفعال التى يقوم بها المستخدم من خلال واجهة التطبيق إلى عمليات التقاط ومعالجة. فعندما يقوم المستخدم بأداء فعل معين باستخدام وحدات الإدخال المتاحة مثل لوحة المفاتيح أو الفأرة فإنه يتم التقاط هذا الفعل من خلال كائن يسمى "كائن مصدر الحدث Event Source Object" أو "كائن النشر Publisher Object" الذى يقوم بعملية الإعلان عن حدوث الفعل للكائن أو الكائنات المقصودة بهذا الفعل ويطلق عليها "الكائنات المستمعة Listener Objects" أو "الكائنات المشاركة Subscriber Objects". يوضح شكل (١٠-٥) مثالاً بسيطاً لمعالجة أحداث واجهة المستخدم. نلاحظ من الشكل أن كل صنف مشارك Subscriber Class يقوم بتطبيق صنف الواجهة Action Listener أو يكون امتداداً لصنف يقوم بتطبيق صنف الواجهة.



شكل (١٠-٥) معالجة أحداث واجهة المستخدم

١٠-٤ الأنماط وواجهات المستخدم Patterns and User Interfaces:

لقد أدى ظهور أسلوب واجهات المستخدم الرسومية GUI إلى اعتماد أسلوب البرمجة المبنية على الأحداث Event-Driven Programming بشكل موسع فى تطوير التطبيقات. وهنا نجد أن مركز التحكم فى البرنامج أو التطبيق أصبح فى يد المستخدم، فهو الذى يحدد ما الذى يجب تنفيذه فى الخطوة التالية من خلال مجموعة الأحداث الموجودة فى التطبيق. كل حدث يتطلب مجموعة من الإجراءات التى تقوم بمعالجته. وقد أدى التوسع فى هذا الأسلوب إلى وجود مجموعة من الأنماط التى يمكن استخدامها فى تصميم واجهات العديد من التطبيقات. سوف نقدم الآن عرضاً لأربعة أنماط تستخدم على نطاق واسع فى تطوير واجهات المستخدم.

١٠-٤-١ نمط المراقب Observer:

هناك العديد من مكونات واجهات المستخدم التى تقوم بمراقبة كائنات التطبيق، مما ينعكس على سلوك واجهة المستخدم بناءً على المعلومات التى تم التقاطها من الكائنات التى حدث بها تغيير. يتم استخدام نمط المراقب فى واجهات المستخدم للتعرف على التغييرات التى تحدث فى مكونات واجهة التطبيق مثل مركب النص Text Box أو

صندوق الاختيار Check Box وغيرها وهى تتطلب تنفيذ إجراء معين يناســب التغيير الذى يحدث مثل اســتدعاء الطريقــة () Text_Changed فى حالة حدوث تغيير فى محتوى مركب النص.

١٠-٤-٢ نمط النقاش Decorator:

يستخدم نمط النقاش لتجميل شكل واجهة المستخدم وتحسين أسلوب استخدامها لتصبح مناسبة للمستخدم بشكل بسيط ومريح User-Friendly. فمثلاً يفضل دائماً اســتخدام مكون الحدود Border لتجميل شــكل النوافذ والإطــارات وبعض مكونات واجهة المســتخدم الأخرى. ولذلك فإن نمط النقاش يســتخدم عادة لتجميل شــكل مكونات واجهة المســتخدم. فيمكننا استخدام مركب المنزلق Scroll Bar لتجميل شكل مكون القائمة JList وتحســين أسلوب اســتخدامها. ويمكننا استخدام مركب شريط القائمــة Menu Bar، وشــريط الأدوات Tool Bar، والحدود Border لتجميل شــكل ظهور مكون الإطار للواجهة Frame.

١٠-٤-٣ نمط سلسلة المسؤولية Chain of Responsibility:

عندما يكون أحد مكونات الواجهة غير قادر على تنفيذ الخدمة المطلوبة الآن، فإنه يقــوم بتمريرها إلى المكونــات الفرعية له، فيقوم أحد المكونات الفرعية بالاســتجابة للطلــب وتنفيذه. أى أن المكــون الأصلى قد فوض أحد المكونات الفرعية له فى تنفيذ الخدمة، ويطلق على هذا الأســلوب اســم "سلسلة المسؤولية" وهو يستخدم على نطاق واسع فى واجهات المستخدم خصوصاً فى نظم المساعدة Help Systems.

١٠-٤-٤ نمط الأمر Command:

يشبه نمط الأمر نمط سلسلة المسؤولية فى المفهوم ولكن يختلف عنه فى الأسلوب. يقوم نمط الأمر بتفويض المكون التالى لتنفيذ الخدمة المطلوبة، فإن لم يستطع فينتقل التفويــض إلى المكــون التالى له حتى يصل إلــى المكون الذى يســتطيع القيام بتنفيذ الخدمــة المطلوبة. أى أن التفويــض فى نمط الأمر يكون محــدداً لمكون معين داخل الواجهة، وذلك بخلاف نمط سلســلة المســؤولية، إذ يكون التفويض غير محدد ويكون مســموحاً لأى مكون فرعى يمكنه تنفيذ الخدمة. يمكن اســتخدام هذا الأسلوب فى تنفيذ عمليات التراجع Undo والإعادة Redo فى واجهات المستخدم.

١٠-٥ واجهات المستخدم على الويب Web-Based User Interface:

من الاتجاهات الحديثـة لتطوير تطبيقات النظم هو تمكينها للعمل من خلال بيئة الإنترنت، حيث يمكن اسـتخدامها فى أى وقت فى أى مكان. وبالفعل فقد قامت هذه التقنية الحديثة بجذب انتباه معظم مطورى النظم للاستفادة من المميزات الكبيرة التى تعود على المؤسسات من اسـتخدامها. فبالإضافة إلى إمكانية استخدام التطبيق من أى مكان وعدم التقيد بمواقع محددة داخل أو خارج المؤسسـة من خلال شبكة محلية LAN، فـإن هذه التقنية تختصر عملية توزيـع وتثبيت برمجيات التطبيق على أجهزة العميـل وتكتفى فقط باسـتخدام برنامج متصفح الويـب Web Browser الذى يقوم بجلـب وعرض معلومـات التطبيق الموجودة فى خادم الويـب Web Server من خلال شبكة الإنترنت. أى أنه يتم توزيع العمل، بحيث تكون قاعدة البيانات وبرامج التطبيق التـى تحتوى على قواعد العمل الخاصة بنظام الأعمـال موجودة على خادم التطبيق علـى الويب Web Server، وتكون البرامج الخاصـة بجلب وعرض معلومات التطبيق موجودة على جهاز العميل Client فى شـكل واجهات المستخدم GUI. تحتوى واجهة التطبيق على مكونات سـاكنة Static باسـتخدام تعليمات HTML ومكونات متحركة Dynamic باستخدام بريمجات جافا Java Applets وشفيرات جافا Java Script.

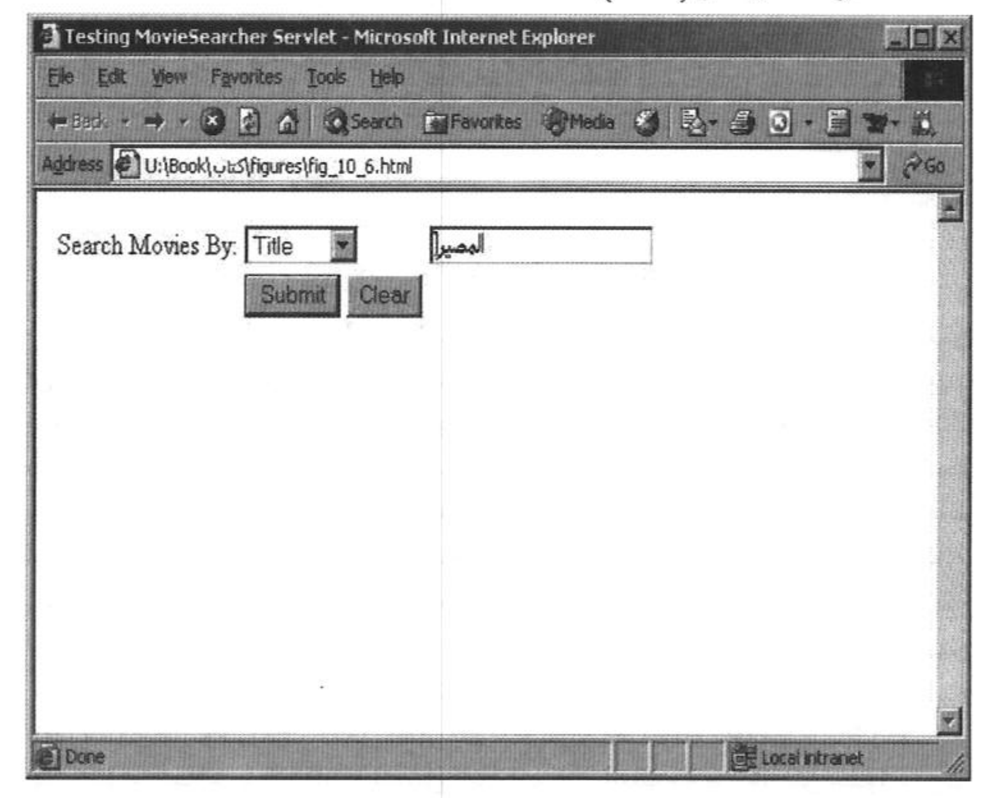
تتسم معظم تطبيقات النظم التى تعمل من خلال شبكة الإنترنت بأنها موزعة على ثـلاث طبقات. الطبقة الأولى هى طبقة العميـل Web Client، والطبقة الثانية هى طبقـة خادم التطبيق Web Server، والطبقـة الثالثة هى طبقة خادم قواعد البيانات Database Server.

١٠-٦ تمكين التقنيات لمستخدمى الويب Enabling Technologies for Web Clients:

تكمـن أهمية أسـلوب تنفيذ التطبيقـات على الويب فى أنها تلبـى رغبات كل من المسـتخدم والمطور للتطبيقات. فمن ناحية المسـتخدم فهو لا يهتم كثيراً بمكان تنفيذ التطبيق سواء كان من جهازه أو من جهاز آخر عبر الإنترنت. وبالنسبة لمهندسى النظم فإن عملية فصل واجهة اسـتخدام التطبيق عن برامج التطبيق وقواعد البيانات تعتبر عملية منطقية، بالإضافة إلى تسهيل عملية نشر مكونات التطبيق على كل من الخادم والعميـل. سـنعرض فى هذا الجزء بعض التقنيات المسـتخدمة فى تصميم واجهات المستخدم على الويب مثل لغة HTML وبعض لغات إعداد صفحات الويب Scripting Languages وبريمجات جافا Java Applets.

١٠-٦-١ لغة HTML:

تعتبـــر لغة Hyper Text Markup Language (HTML) هى أكثر اللغات شــيوعاً فى تصميم صفحات الويب. تأخذ صفحات HTML شــكلاً قياســياً بسيطاً يتضمن مجموعة من العلامات Tags التى تسمح بظهور الصفحات بشكل جيد على متصفحات الويــب Web Browsers. يوضــح شــكل (١٠-٦) مثالاً على اســتخدام لغة HTML لتصميم صفحة واجهة اســتخدام تقوم بالاســتعلام عن الأفلام فــى قاعدة بيانات الأفــلام Movie Actor. ويتضــح من الصفحة أن المســتخدم يقوم باختيار أســلوب البحــث مثــل العنوان Title، ثم يقوم بإدخال اســم الفيلم ليتــم تنفيذ البحث وعرض تفاصيل الفيلم. ويوضح شكل (١٠-٧) محتويات ملف HTML الذى يقوم بإنتاج هذه الصفحة الموضحة بشكل (١٠-٦).



شكل (١٠-٦) مثال لصفحة HTML

Sample HTML document

```
<html>
<head>
<!-- comment: مثال لإظهار عنوان لصفحة ويب --!>
<title>  Testing MovieSearcher Servlet  </title>
<body>
        <form  method="post"  action="/SearchMovie">
        <table  border=0  width="70%">
            <tr><td>Search Movies By:</td>
                <td><select name="searchBy">
                    <OPTION  value="TitleV">Title
                    <OPTION  value="ActorV">Actor
                    <OPTION  value="DirectorV">Director
                </select></td>
                <td><input  name=searchField  type=text></td>
            </tr>
            <tr><td></td><td><input  type="submit"  value="Submit">
            <input  type=reset  value="Clear"</td></tr>
       </table>
       </form>
 </body>
 </html>
```

شكل (١٠-٧) محتويات ملف HTML

يبدأ دائماً ملــف HTML بالعلامة <html> وينتهــى بالعلامة </html>. ويمكن أن يتضمــن الملف جــزءاً خاصاً فى المقدمة <head> يســتخدم لإعلام متصفح الويب بمعلومات عن الملف نفســه. ويمكن أيضاً اســتخدام الجمل التوضيحية Comments فى أى مكان فى الملف باســتخدام العلامات <-- ، --!>. يتم إضافة عنوان الصفحة فى مقدمة الملف باســتخدام العلامــة </title> ، <title>. نلاحظ أن معظم علامات لغــة HTML تحتاج إلى إغلاق باســتخدام الرمز / قبل اســم العلامة. ولكن هناك بعض العلامات التى لا تحتاج إلى إغلاق مثل <option>. ونلاحظ أيضاً من الشــكل أن بعض العلامات يكون لها معاملات أو خصائص مثل: علامة <form> التى تتضمن خاصيتين هما (method, action). تســتخدم العلامة <body> و </body> لتوضيح الجزء الذى ســوف يظهر داخل إطار متصفح الويب عند تنفيذ صفحة HTML. يتم اســتخدام العلامــة <table> لتمثيل البيانات على الصفحة فى شــكل جدول. ويتم تنظيم البيانات داخل الجدول عن طريق الصفوف <tr> والأعمدة <td>.

يتم فى هذا المثال استقبال المدخلات من المستخدم داخل صفحة المتصفح باستخدام العلامة <form> وهى تتضمن طريقة من النوع "post" وهى تعنى أنه سيتم إرسال البيانات المجمعة داخل نموذج الصفحة إلى الخادم أو مصدر خارجى للمعلومات مثل قواعد البيانات. يتم تحديد عنوان الخادم Server URL من خلال المعامل action للعلامة <form>. يقوم المستخدم باختيار أحد بيانات القائمة باستخدام العلامة <select> وإدخال بيانات فى مركب النص text باستخدام العلامة <input>. هناك نوع آخر من الإدخال وهو <submit> هو يمثل بمفتاح إدخال button يقوم بإحالة البيانات الموجودة بالنموذج <form> إلى عنوان الخادم. وهناك أيضاً النوع <reset> الذى يستخدم عادة لإعادة محتويات النموذج إلى حالتها الأصلية (حقول فارغة أو قوائم ذات اختيار مبدئى أو افتراضى). بالطبع تحتوى لغة HTML على مجموعة كبيرة من العلامات التى تستخدم لتصميم صفحات الويب ولكن لا يتسع المجال هنا لعرضها جميعاً، ولكن نكتفى بعرض بعض العلامات التى نستخدمها فى هذا المثال التوضيحى.

١٠-٦-٢ لغات إعداد صفحات الويب Scripting Languages:

تستخدم لغة HTML بعض الأدوات البرمجية لدعم عمليات التعامل مع المستخدم من خلال متصفح الويب. تعتبر لغات VB Script, Java Script من تلك الأدوات المستخدمة على نطاق واسع فى ملفات HTML لإجراء عمليات التحقق السريعة للمدخلات على الصفحة ولتشغيل الصور المتحركة Animation والعمليات الأخرى التى تضفى صفة الحركة على صفحة الويب. يجب تعريف نوع اللغة المستخدمة فى مقدمة ملف HTML. يوضح شكل (١٠-٨) مثالاً لاستخدام لغة Java Script للتحقق من صحة البيانات المدخلة عن طريق المستخدم (سطر ٢٢) والتأكد من أنها تقع داخل المدى المسموح (سطر٢٠). يتم تنفيذ تعليمات التحقق عندما يقوم المستخدم بالضغط على مفتاح Submit، حيث يتم استدعاء الدالة () Checkit من خلال المعامل Action المصاحب للعلامة <Form> (سطر٢٠). تقوم الدالة () Checkit (سطر ٥) بالتحقق من القيمة المدخلة (سطر ٢٢) للتأكد من وقوعها ضمن المدى المعرف بالقيمة الصغرى Lower والقيمة الكبرى Upper. فإذا كانت القيمة تقع ضمن المدى المحدد تسترجع الدالة القيمة True وغير ذلك تسترجع القيمة False مع إظهار رسالة تشير إلى عدم صحة البيانات المدخلة (سطر ١١،١٢).

Sample Java Script

```
<HEAD>
        <TITLE> Form Validation </TITLE>
        <SCRIPT LANGUAGE="JavaScript">
          <!—
                function checkIt(int lower, int upper) {
                    var strval = document.sampleForm.percent.value;
                    var intval = parseInt(strval);
                    if ( lower < intval && intval < upper ) {
                        return (true) ;
                    } else {
                      alert("Input " + strval + " is out of "+ lower +
                            "-" + upper + " range") ;
                      return (false) ;
                    }
                }
            <!-- end script -->
        </SCRIPT>
</HEAD>
<FORM NAME="sampleForm" METHOD="post"
        ACTION="/SearchMovie" onSubmit="checkIt(0,100)">
        <P>Percentage given:
        <INPUT TYPE="text" NAME="percent" VALUE="1">
        <BR><INPUT TYPE="submit">
</FORM>
```

شكل (١٠-٨) مثال على استخدام لغة Java Script **داخل ملف** HTML

١٠-٦-٣ البريمجات Applets:

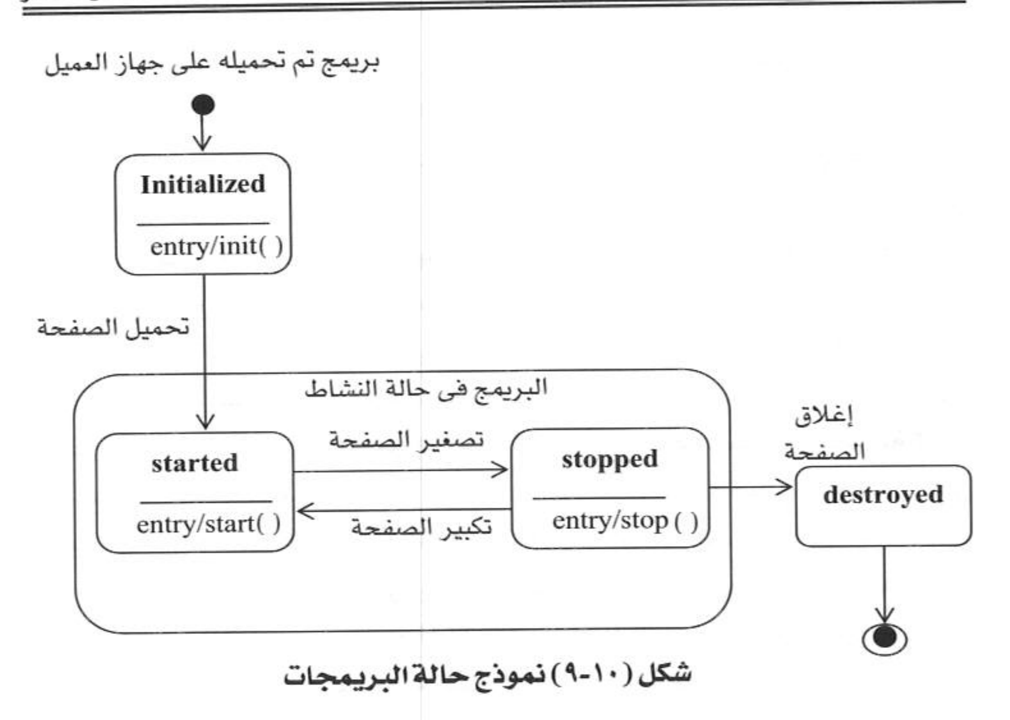
البريمــج Applet هو برنامــج صغير يؤدى وظيفة معينة ويكــون مخزناً فى خادم الويب ولكن يتم تحميله وتنفيذه من خلال متصفح الويب Web Browser عند العميل. هناك نوعان من البريمجات: بريمج خفيف يسمى Thin Applet، وبريمج ثقيل يسمى Thick Applet. يتفــق النوعان فى أنهما يحتويان علــى مجموعة من الأصناف التى يتضمنها التطبيق ويختلفان فى حجم الكود الذى يتم تنفيذه من خلال المتصفح. ففى حالــة البريمــج الخفيف يتم تنفيذ أغلب الوظائف من علــى الخادم ويتم تحميل جزء بســيط فقط إلى جهاز العميل ليتم تنفيذه من خلال المتصفح. وهو يتميز بالســرعة

أثناء التنفيذ نظراً لصغر حجمه وهو يناسـب الشبكات البطيئة، ولكنه يكون صعباً فى الصيانة وتتبع الأخطاء التى تحدث فى أثناء التنفيذ نظراً لاعتماده على الخادم بشكل كبير. أما فى حالة البريمج الثقيل فإنه يتم تحميل جزء كبير من البرنامج الموجود على الخادم إلى جهاز العميل ويتم تنفيذه من خلال المتصفح، وهو يناسـب الشـبكات ذات السرعات العالية وتكون صيانته أسهل نظراً لعدم اعتماده على الخادم بشكل كبير فى أثناء التنفيذ. يوضح شكل (١٠-٩) نموذجاً لحالة البريمج حيث إنه يمر بأربع حالات. جدير بالذكر أن عملية إنشـاء البريمج Applet Construction تتم قبل اسـتدعائه، وبعد ذلك يقوم المتصفح باسـتدعاء الطريقـة init () لتهيئة الكائن (البريمج) لعملية التحميل. بعد ذلك يدخل البريمج فى حالة النشـاط وذلك باستدعاء الطريقة start لبدء عملية التنفيذ، ويظل فى هذه الحالة طالما كانت صفحة المتصفح مفتوحة. ينتقل البريمـج إلى حالة التوقـف stop () فى حالة أن يكون غير نشـط (فى أثناء تصغير نافـذة المتصفح) وذلك لتوفير الموارد التى يسـتخدمها البريمـج. وعندما يتم إغلاق صفحة الويب ينتقل البريمج إلى حالة الانتهاء destroy () ويتم تحرير الموارد التى كان يسـتخدمها البريمج وإعادتها إلى حالتها الأولية. يوضح شـكل (١٠-١٠) مثالاً على استخدام البريمجات داخل صفحة HTML على متصفح الويب.

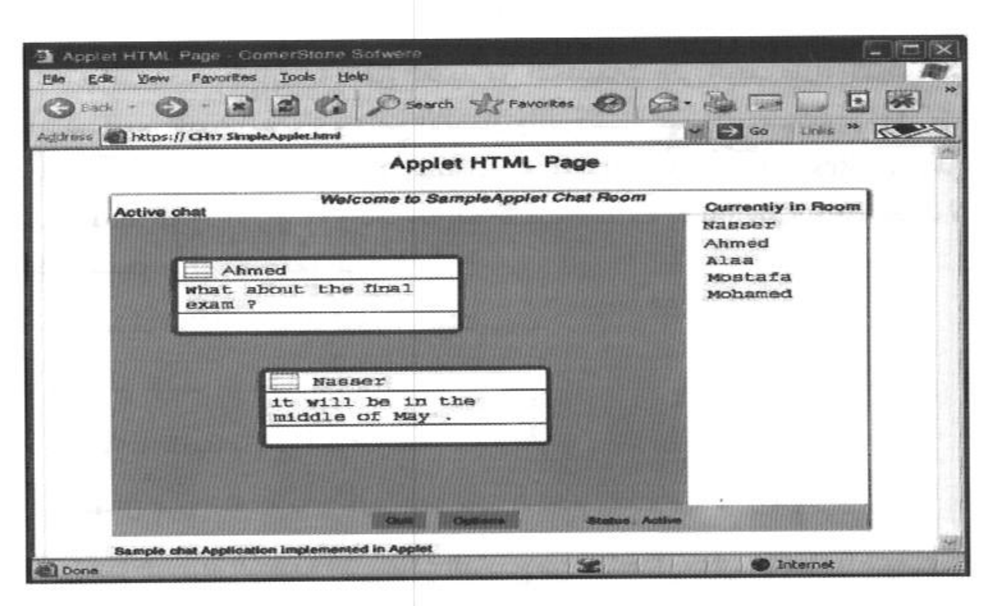
يتم اسـتخدام البريمجات فى العديد من التطبيقات على الويب مثل: التطبيقات البنكيـة ومراقبـة المخزون وبرامج الاتصال والدردشـة عبر الإنترنـت وغيرها. يتم عادة تشـغيل البريمجات من خلال بيئة مقيدة Restrictive Environment تمنعه من اسـتخدام الموارد الأساسية والحرجة للحاسـب المضيف Host Computer بحيث لا يكون مسموحاً للبريمج بما يلى:

- القراءة من ملفات النظام للجهاز المستضيف.
- كتابة أو حذف ملفات منه.
- تحميل أو تنفيذ برامج أخرى.
- الاتصال عبر الشبكة بأى جهاز آخر غير الخادم القادم منه البريمج.

وتعتبر الوسيلة الوحيدة التى تمكن البريمج من استخدام موارد الجهاز المضيف هى أن يقوم المطور بمنح تلك الصلاحية للبريمج.



شكل (١٠-٩) نموذج حالة البريمجات



شكل (١٠-١٠) استخدام البريمجات

١٠-٧ تمكين التقنيات لخادم الويب Enabling Technologies for Web Server:

يقــوم خادم الويب بإدارة عملية الاتصال بين عملاء الويب والتطبيق الخاص بنظام الأعمال. يتم عادة تطبيق عمليات نظام الأعمال داخل خادم الويب أو من خلال خادم خاص بالتطبيقات Application Server. يقوم خادم الويب أيضاً بالوظائف التالية:

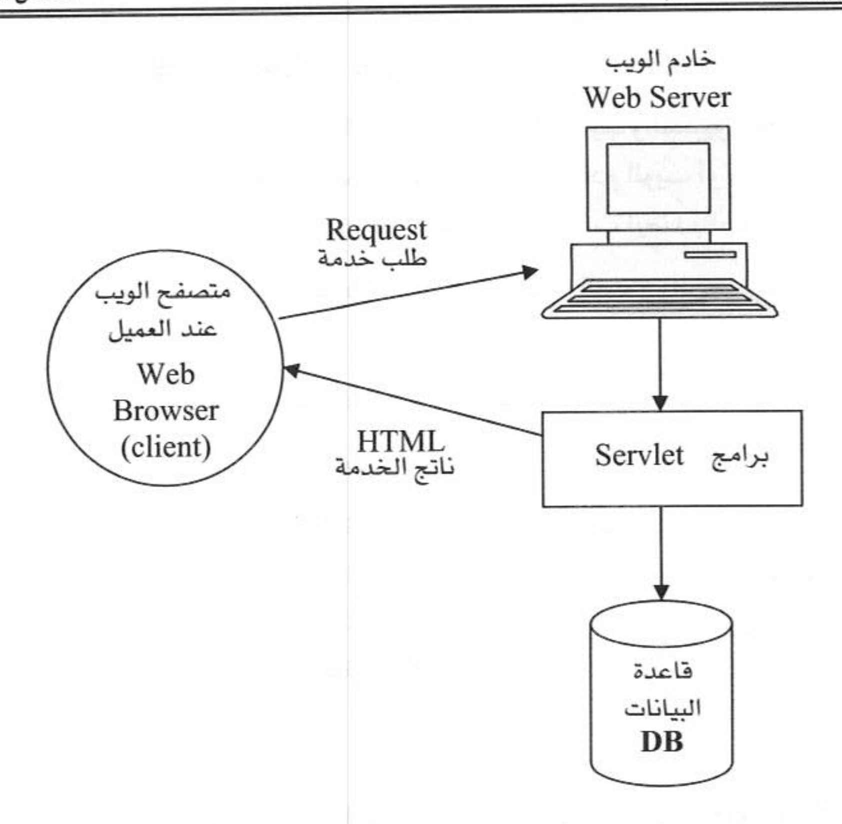
- تقــديم المحتــوى الديناميكى لصفحات الويــب مثل: الصور والرســومات وملفات الصوت، وغيرها.
- التحكم فى تدفق الشاشات من خلال الروابط فى متصفح العميل.
- صيانة الحالات الإجرائية عند العميل.

يتوافــر الآن العديــد مــن التقنيات التــى تدعــم عمليات خــادم الـويب مثل: Common Gateway Interface (CGI) وهى كانت تستخدم فى الفترة السابقة ولكن تم اســتحداث أســلوبين آخرين أكثر مقدرة منها هما Servlets, JSP عن طريق شركة Sun Micro Systems من خلال تقنية J2EE.

١٠-٧-١ تقنية Servlet:

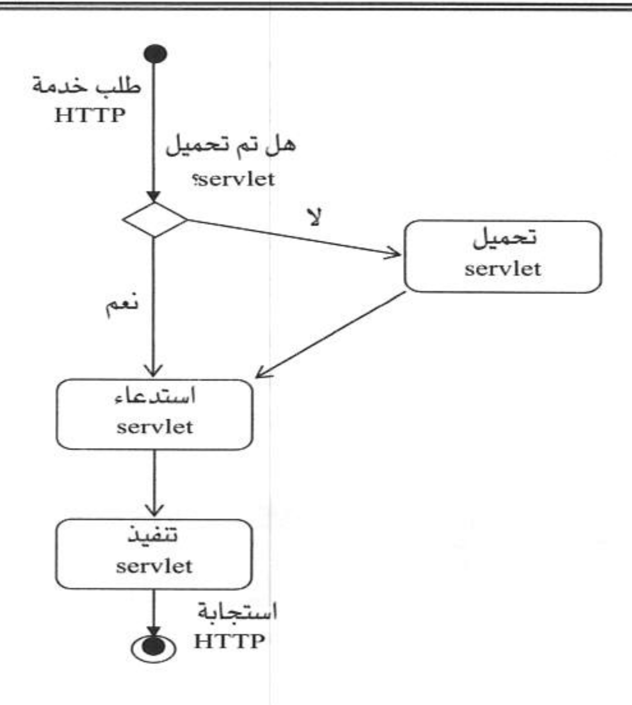
يعتبر الـ Servlet هو برنامج جافا يتم تخزينه على خادم الويب Web Server ويقوم بتقــديم خدمة معينة للعديــد من أجهزة العمــلاء Clients المتصلة بالخادم عن طريق شــبكة الإنترنت. ويتضح الفرق بين Servlet والبريمــج Applet فى أن Servlet يكون موجــوداً دائماً على الخادم ويتم ترجمته وتنفيــذه أيضاً من على الخادم. أما البريمج فيتم تحميله إلى جهاز العميل وتنفيذه من خلاله. يوضح شكل (١٠-١١) كيفية الاتصال بين أجهزة العميل وخادم الويب لتنفيذ خدمة معينة باستخدام تقنية Servlet.

يحتوى خــادم الويب على برامج Servlets التى تقــدم الخدمات المطلوبة للعملاء، ويقــوم خادم الويب باســتقبال طلب الخدمة من جهاز العميــل (يكون غالباً من خلال صفحة HTML أو صفحة JSP) ثم يقوم بالبحث عن Servlet المناسب لتنفيذ الخدمة المطلوبة ويتم تنفيذها من على الخادم واســترجاع النتائــج للعميل طالب الخدمة فى شــكل HTML. يقوم Servlet فى أثناء التنفيذ باستخدام موارد الخادم ومن ضمنها قواعد البيانات.



شكل (١٠-١١) الاتصال بين العميل وخادم الويب فى تقنية Servlet

يدخل Servlet فى حالة النشاط عندما يتم استدعاؤه من الخادم باستخدام الطريقة () init ويظل فى هذه الحالة عند وجود طلبات من العملاء من خلال برتوكول HTTP. وعندما تنتهى طلبات العملاء يتم استدعاء الطريقة () destroy لإنهاء Servlet وتحرير الموارد التى كان يستخدمها. يوضح شكل (١٠-١٢) كيفية استدعاء Servlet من خلال طلبات HTTP، حيث يقوم الخادم بالبحث عنه والتأكد من أنه موجود بالذاكرة أم لا، فإذا لم يكن موجوداً يتم تحميله أولاً ثم استدعاؤه من خلال الطريقة () init ويتم تنفيذه واسترجاع ناتج الخدمة إلى العميل من خلال HTTP Response فى شكل HTML. ويوضح شكل (١٠-١٣) مثالاً على استخدام تقنية Servlet لإظهار معلومات مستخرجة من قاعدة بيانات الأفلام الموجودة على الخادم وذلك نتيجة لطلب استدعاء هذه المعلومات من جهاز أحد العملاء.



شكل (١٠-١٢) خطوات استدعاء وتنفيذ Servlet

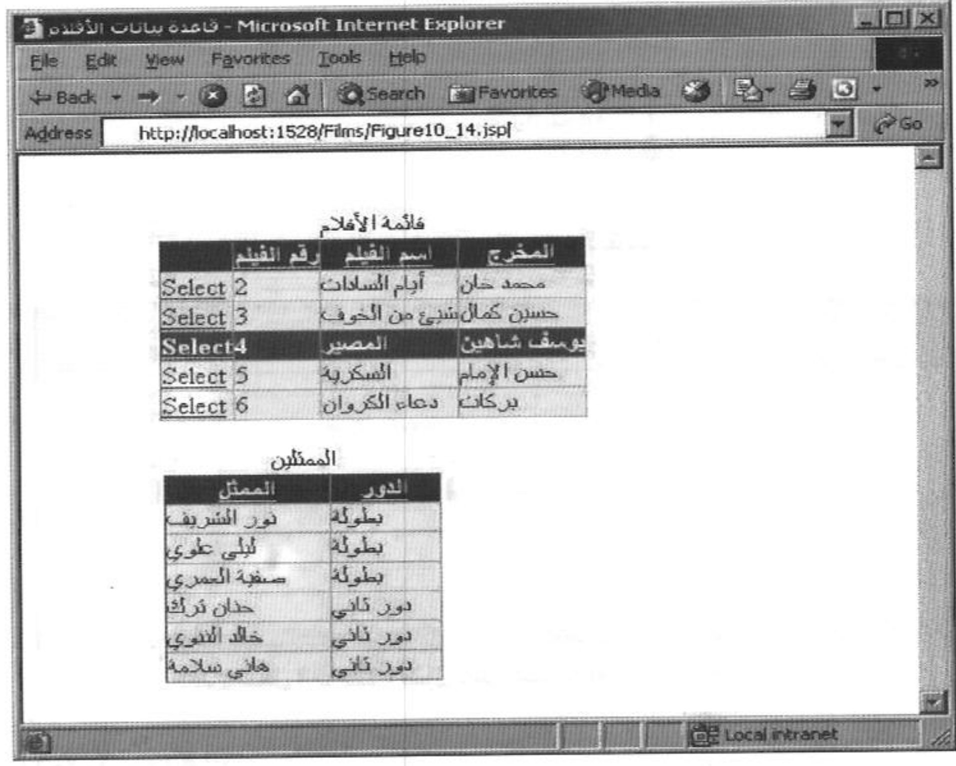


شكل (١٠-١٣) مثال على استخدام ال Servlet

١٠-٧-٢ تقنية JSP:

تعتبــر تقنية إعداد صفحات خــادم الويب بلغة جافــا (Java Server Pages (JSP أسلوباً مشابهاً لأسلوب Servlet من حيث الاتصال بين العميل والخادم كما هو موضح

فى الشكل (١٠-١١) ولكنه يختلف عنه من ناحية فصل منطقة العمل Business Logic عن منطقة العرض Presentation؛ حيث يتم تمثيل منطقة العمل باستخدام Servlet وتمثيل منطقة العرض باستخدام صفحات JSP. ولهذا فإن أسلوب JSP يحتاج إلى خطوتين إضافيتين قبل عملية استدعاء Servlet. يجب تحليل محتويات صفحة JSP أولاً وإنتاج الكود الذى سيتم ترجمته وتجهيزه للاستخدام، وبعد ذلك تتم عملية استدعاء servlet وتنفيذه. أى أنه يمكننا القول إن الجزء الديناميكى لمتطلبات العميل يتم تطبيقه باستخدام Servlet وتضمينه داخل صفحة JSP التى تحتوى على تعليمات عن كيفية عرض المخرجات عند العميل. يوضح شكل (١٠-١٤) مثالاً لاستخدام تقنية JSP لعرض قائمة الأفلام بالإضافة إلى عرض قائمة الممثلين المشاركين فى الفيلم المحدد بقائمة الأفلام.



شكل (١٠-١٤) مثال لاستخدام تقنية JSP

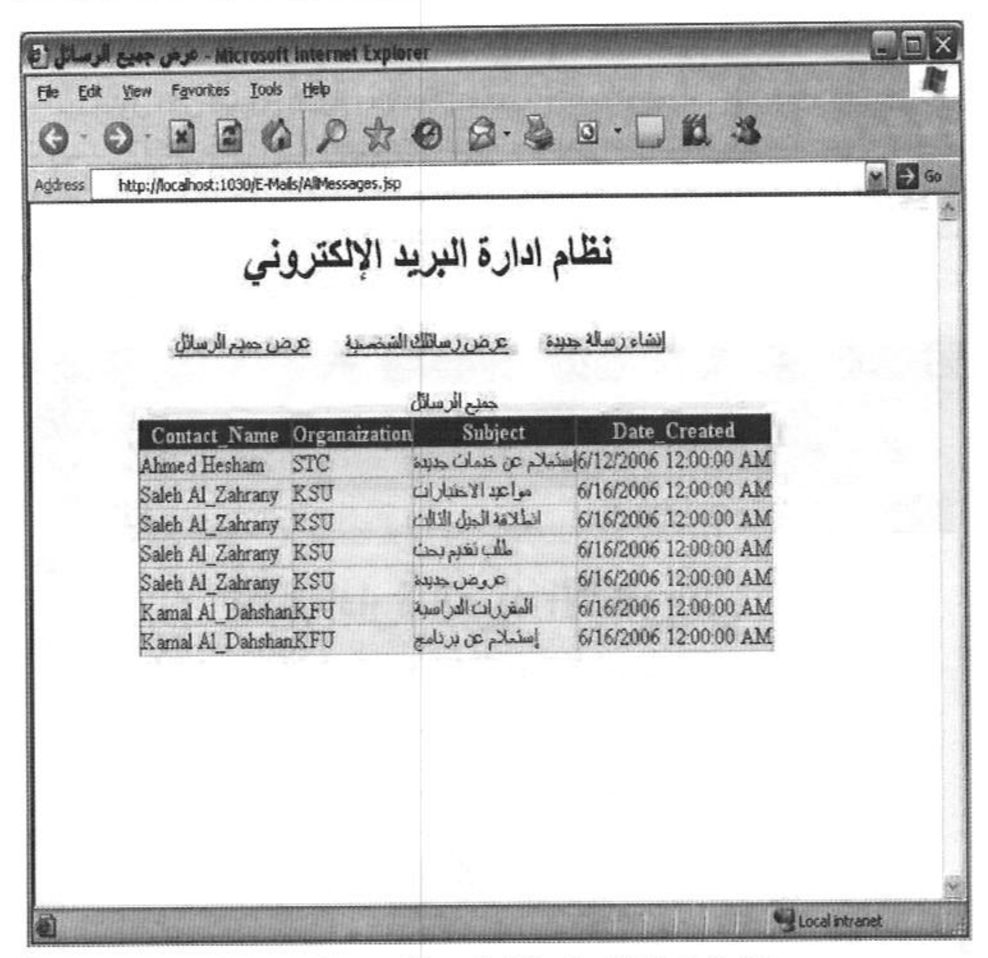
١٠-٨ تطبيق على واجهات المستخدم User Interface Application:

سنحاول الآن تطبيق تقنية Servlet لتصميم وتنفيذ واجهات استخدام تطبيق إدارة البريد الإلكتروني EM، بحيث يتم تنفيذه بالكامل من خلال الويب. يبدأ التطبيق بعرض شاشة الدخول للمستخدم Login باستخدام لغة HTML كما هو موضح بالشكل (١٠-١٥). يتم استدعاء Servlet لتنفيذ عملية التحقق من بيانات المستخدم عندما يتم الضغط على مفتاح "دخول".



شكل (١٠-١٥) شاشة الدخول للتطبيق EM

وبعد نجاح عملية الدخول يتم عرض الصفحة الرئيسية للمستخدم وهى تحتوى على مجموعة الرسائل الخاصة بالمستخدم ولم يتم إرسالها كما هو موضح بالشكل (١٠-١٦). ومن خلال هذه الصفحة يمكن للمستخدم أن يستعرض محتويات أى رسالة وأن يقوم بإرسالها. جدير بالذكر أن Servlet نفسه المستخدم فى تنفيذ عملية التحقق من بيانات المستخدم هو الذى يقوم الآن بتنفيذ خدمات هذه الصفحة أيضاً.



Contact_Name	Organaization	Subject	Date_Created
Ahmed Hesham	STC	إستعلام عن خدمات جديدة	6/12/2006 12:00:00 AM
Saleh Al_Zahrany	KSU	مواعيد الاختبارات	6/16/2006 12:00:00 AM
Saleh Al_Zahrany	KSU	انطلاقة الجيل الثالث	6/16/2006 12:00:00 AM
Saleh Al_Zahrany	KSU	طلب تقديم بحث	6/16/2006 12:00:00 AM
Saleh Al_Zahrany	KSU	عروض جديدة	6/16/2006 12:00:00 AM
Kamal Al_Dahshan	KFU	المقررات الدراسية	6/16/2006 12:00:00 AM
Kamal Al_Dahshan	KFU	إستعلام عن برنامج	6/16/2006 12:00:00 AM

شكل (١٠-١٦) الصفحة الرئيسية لبريد المستخدم

وعندما يقوم المستخدم باختيار إحدى الرسائل غير المرسلة ويضغط على مفتاح select فإنه يتم عرض محتويات الرسالة فى الجزء العلوى من الصفحة وهو يتضمن أيضاً مفتاح Modify لإمكانية التعديل فى محتوى الرسالة كما هو موضح بالشكل (١٠-١٧).

وعندما يضغط المستخدم على مفتاح Modify يتم الانتقال إلى صفحة تعديل محتوى الرسالة وهى تحتوى على جميع خصائص الرسالة ومحتوياتها الحالية مع إمكانية التعديل فى أى خاصية منها كما هو موضح بالشكل (١٠-١٨). وبعد الانتهاء

مــن إجراء التعديل على الرســالة يتم الضغط على مفتــاح Update، فيقوم Servlet بتنفيذ التعديلات المطلوبة عند خادم الويب.



شكل (١٠-١٧) عرض محتويات الرسائل

وعندما يطلب المســتخدم إنشاء رسالة جديدة من خلال الصفحة الرئيسية (شكل (١٠-١٦)) فإنه يتم الانتقال إلى الصفحة الموضحة بالشكل (١٠-١٩) وهى تحتوى على جميع خصائص الرســالة، بعضها متروك خالى للتعبئة من قبل المســتخدم، والبعض الآخر يحتوى على معلومات افتراضية مثل: تاريخ الإنشــاء واســم المســتخدم. وبعد الانتهاء من تعبئة الرســالة الجديدة يقوم المستخدم بالضغط على مفتاح save لحفظ الرسالة.

تعديل على رسالة - Microsoft Internet Explorer

نظام البريد الإلكتروني

Edit Message No. : 1

Subject : إستعلام عن خدمات جديدة

Creator Name : Ahmed Al_Zahrany

Creation Date : 12/6/2006

Assigned To : Nasser Al_Meshary

Addressed To : Saleh Al_Zahrany

Schedule Date :

Message Body : الرجاء التكرم بموافاتنا بمعلومات

Update

شكل (١٠-١٨) تعديل محتويات الرسالة

رسالة جديدة - Microsoft Internet Explorer

http://localhost:1030/E-Mails/NewMessage.jsp

نظام البريد الإلكتروني

Message No. : 8

Subject :

Creator Name : Ahmed Al_Ghamdy

Creation Date : 6/17/2006

Assigned To : Ahmed Al_Ghamdy

Addressed To : Ahmed Hesham

Schedule Date :

Message Body :

Save

شكل (١٠-١٩) إنشاء رسالة جديدة

المراجع

أولاً - المراجع الأجنبية:

1. Sommerville I. ، "Software Engineering 7 ".7th Edition، Pearson، Addison-Wesley، Harlow، Essex، UK، 2004.
2. Pressman ،"Software Engineering: A Practitioner's Approach"، 5th edition، McGraw-Hill، 2001.
3. Leszek Maciaszek and Bruc Liong ، "Practical Softwrae Engineering: a case study approach"، Addison-Wesley، 2005.
4. J. Rumbaugh، I. Jacobson، and G. Booch ،"The Unified Modeling Language Reference Manual"، second edition، Addison Wesley، 2005.
5. Dwayne Phillips، "The Software Project Manager's Handbook"، second edition، Wiley-Interscience، 2004.
6. Frank Tsui "Managing Software Projects"، Jones and Bartlett، 2004.
7. James Taylor، "Managing Information Technology Projects"، AMACOM، 2004.
8. Ghezzi، Jazayeri، and Mandrioli "Fundamentals of Software Engineering"، Prentice Hall، 2003.
9. Joseph Schmuller، "Teach Yourself UML"، SAMS، 2004.
10. Martin Fowler ، "UML Distilled"، third edition، Addison Wesley، 2004.
11. J. George، D. Batra، J. Valacich، and J. Hoffer، "Object-Oriented System Analysis and Design"، Prentice Hall، 2004.
12. Dennis Lock "Project Management"، eighth edition، Gower، England، 2003.
13. Sun Microsystems، "Java Programming language"، Course S-110، sun Educational Services، 2003.

14. Sun Microsystems، "Java Programming language"، Course SL-275، sun Educational Services، 2003

15. Eve Andersson، Philip Greenspun، and Andrew Grumt ،"Software Engineering for Internet applications"، MIT Press ، 2003.

16. Stephen R. Schach، "Object-Oriented and Classical Software Engineering"، 5th Edition، McGraw-Hill، 2002.

17. J. George، J. Valacich، and J. Hoffer، "Modern System Analysis and Design"، third edition، Prentice Hall، 2002.

18. J. Hoffer، M. Prescott، and F. McFadden، "Modern Database Management"، sixth edition، Prentice Hall، 2002.

19. T. Connolly، and C. Begg، "Database Systems: A Practical approach to Design، Implementation، and Management" third edition، Addison Wesley، 2002.

20. Herbert Scildt ،"The Complete Reference، JAVA2"، fifth edition، Osborne، 2002..

21. Deitel و Deitel، "JAVA How to Program"، 4th Edition، Prentice Hall، 2002.

22. Savitch، "JAVA: An Introduction to computer Science & Programming"، Second Edition، Prentice Hall، 2002..

23. Stevens P. with Pooley، R. "Using UML: Software Engineering with Objects and Components"، Addison-Wesley، Harlow، Essex، UK، 2000.

24. Gamma، Helm، Johnson، and Vlissides "Design Patterns: Elements of Reusable Object-Oriented Software"، MA، Addison-Wesley، 1995.

25. eProject(2006) ، http://www.eproject.com.

26. eRoom(2006)، http://www.eroom.net/eroomnet/

27. ManagePro(2006)، http://www.managepro.net/

28. Microsoft(2006)، http://www.microsoft.com.office.project/default.asp

29. Oracle(2006)، http://www.oracle.com/products/jdev/content.html

30. Palisade(2006)، http://www.palisade-europe.com

31. Primavera(2006). http://www.primavera.com

32. Rational(2006). http://www.rational.com

33. Sun(2006). J2EE platform. available at http://www.java.sun.com

34. Sun ONE(2006). http://www.sun.com/software/sundev/jde/

35. Sybase(2006). http://www.sybase.com/products/enterprisemodeling

ثانياً - المراجع العربية:

١- د. عبدالوهاب السماك، "إدارة مشاريع البرمجيات"، معهد الإدارة العامة، الرياض، ٢٠٠٥.

٢- د. عبدالوهاب السماك، "تخطيط وتنفيذ مشاريع مراكز الحاسب الآلى"، معهد الإدارة العامة، الرياض، ٢٠٠٤.

٣- د. عبدالوهاب السماك، "تحليل النظم باستخدام لغة UML"، معهد الإدارة العامة، الرياض، ٢٠٠٥.

٤- د. عبدالوهاب السماك، "تحليل النظم الهيكلية"، معهد الإدارة العامة، الرياض، ٢٠٠٣.

٥- د. عبدالوهاب السماك، "البرمجة بلغة جافا"، معهد الإدارة العامة، الرياض، ٢٠٠٤.

٦- محمد رسمى، "تصميم النظم باستخدام لغة UML"، معهد الإدارة العامة، الرياض، ٢٠٠٥.

٧- عبدالحميد بسيونى، "أساسيات هندسة البرمجيات"، دار الكتب العلمية للنشر والتوزيع، القاهرة ، ٢٠٠٥.

٨- د. نزار الحافظ، د. ندى غنيم، د. غيداء ربداوى، "هندسة البرمجيات: منهج للممارس" مطبوعات الجمعية العلمية السورية للمعلوماتية، دمشق، ٢٠٠٢.

المؤلف فى سطور

د. **عبدالوهاب كامل محمد السماك.**

من مواليد القاهرة – جمهورية مصر العربية.

المؤهل العلمى:

دكتوراه الفلسفة فى هندسة الحاسبات، عام ١٩٩٢م من جامعة الزقازيق بجمهورية مصر العربية.

الوظيفة الحالية:

أستاذ الحاسبات المساعد بمعهد الإدارة العامة، الرياض.

الخبرات العلمية والعملية:

- له العديد من الأبحاث العلمية المنشورة فى المؤتمرات والدوريات العلمية المتخصصة فى هندسة الحاسبات.
- التدريس فى عدد من الجامعات المصرية (الحكومية، الأهلية).
- المشاركة فى مشروع "تطوير التعليم الهندسى" فى مجال هندسة الحاسبات بجمهورية مصر العربية.
- المشاركة فى مشروع "اللغويات الحسابية العربية" التابع لأكاديمية البحث العلمى بجمهورية مصر العربية.
- عضو لجنة تطوير دبلوم برمجة الحاسب فى قطاع الحاسب الآلى بمعهد الإدارة العامة.
- إعداد مجموعة من الحقائب التدريبية فى مجالات تحليل وتصميم النظم والبرمجة وإدارة مشاريع البرمجيات لقطاع الحاسب الآلى بمعهد الإدارة العامة.
- المراجعة العلمية لبعض الكتب المنشورة فى مطبوعات معهد الإدارة العامة فى مجال الحاسب الآلى.
- الإشراف على مشاريع التخرج لطلبة دبلوم برمجة الحاسب بمعهد الإدارة العامة.

تم التصميم والإخراج الفنى والطباعة فى
الإدارة العامة للطباعة والنشر بمعهد الإدارة العامة - ١٤٢٩هـ

هذا الكتاب :

تشهد معظم البلدان العربية الآن طفرة كبيرة في استخدام تقنية المعلومات فى المجالات المختلفة فى محاولات جادة لتطبيق إستراتيجيات الحكومة الإلكترونية من خلال تطوير تطبيقات الأعمال الخاصة بكل مجال.

تتسم معظم تطبيقات الأعمال بكبر حجمها واحتوائها على قدر كبير من التعقيد. ومن ثم فهى تحتاج إلى تطوير برمجيات تتميز بالجودة والمرونة وسهولة الصيانة.

ومع ازدياد اعتماد المؤسسات والهيئات والشركات على استخدام تقنية المعلومات فى إدارة وتشغيل الأنشطة الخاصة بها، فقد أدى ذلك إلى زيادة فى متطلبات ومواصفات البرمجيات لكى تكون أكثر سرعة ودقة ومرونة وأمناً وقابلة للاتساع والتحديث. بالإضافة إلى إمكانيات التعامل مع تطبيقات أخرى والعمل فى بيئات تشغيل مختلفة.

إن عملية تطوير البرمجيات لتلك التطبيقات تعتبر مكلفة وتستغرق وقتاً طويلاً. ويقوم على تنفيذها فرق عمل مختلفة. لذا فإنها تحتاج إلى إدارة جيدة لكى تمكنها من تحقيق أهدافها بدقة. ويتطلب ذلك استخدام مجموعة من الأساليب والأدوات الحديثة التى تدعم عمليات تطوير البرمجيات بشكل جيد.

وسوف نركز فى هذا الكتاب على أسلوب تطوير البرمجيات المبنى على الكائنات ليلائم مشاريع البرمجيات الكبيرة الحجم والقابلة للاتساع. وسوف نستعين ببعض التطبيقات والحالات الدراسية لتعزيز عملية فهم كيفية استخدام تلك الأساليب.

ردمك: ٥-١٦٩-١٤-٩٩٦٠-٩٧٨

تصميم وإخراج وطباعة الإدارة العامة للطباعة والنشر - معهد الإدارة العامة ١٤٢٩هـ